DESARROLLO DE UN PROCEDIMIENTO BASADO EN UNA METODOLOGÍA ÁGIL PARA LA CONSTRUCCIÓN DE LA APLICACIÓN MÓVIL DEL SISTEMA DE PETICIONES, QUEJAS, RECLAMOS, DENUNCIAS Y SUGERENCIAS (PQRDS) DE LA UNIVERSIDAD DE PAMPLONA

AUTOR:
JESÚS MIGUEL SIERRA VÁSQUEZ

DIRECTOR:
EDGAR ALEXIS ALBORNOZ ESPINEL
Ingeniero de Sistemas

UNIVERSIDAD DE PAMPLONA
FACULTAD DE INGENIERÍAS Y ARQUITECTURA
DEPARTAMENTO DE INGENIERÍAS ELECTRÓNICA, ELÉCTRICA, SISTEMAS Y TELECOMUNICACIONES
PROGRAMA DE INGENIERÍA DE SISTEMAS
PAMPLONA, NORTE DE SANTANDER – COLOMBIA
2015

DEDICATORIA

A Dios; a mis amados padres: Nelvina Vásquez y Hugues Sierra; a mi querida familia: Sandra Mileth, Hugues Miguel, Miguel Fernando, Nick Brandon, María Salem, Isaac Miguel y Taliana.

AGRADECIMIENTOS

Agradezco principalmente a Dios quien fue siempre mi apoyo y fortaleza en los momentos que veía imposible completar este proyecto, a mi familia por su respaldo constante. A Jesús Ángulo y Alfredo Guzmán quienes fueron mis compañeros de estudio durante la mayor parte de mi carrera universitaria.

Por otro lado, expresar mi gratitud al docente Edgar Albornoz quién ha sido fundamental con su enseñanza en mi proceso de aprendizaje en el inicio y final de mi carrera universitaria, y que como director de este proyecto me ha guiado, apoyado y corregido durante todo el proceso de elaboración de este. También agradecer a los docentes Mauricio Rojas, Luis Esteban, Sergio Peñaloza, Orlando Maldonado, Ailin Orjuela, entre otros, que contribuyeron en todo mi proceso de aprendizaje. Y un especial agradecimiento a la docente Laura Villamizar quien ha marcado un aspecto importante de mi vida, induciéndome por el camino de la investigación.

Asimismo, agradecer a mis compañeros de oficina Sandra Mantilla Durán, Roger Legizamón Céspedes, Andrés Villamizar Vera, Jesús Castellanos Guerrero y el coordinador técnico de desarrollo Wilfred Villalba Montagut, quienes hicieron más fácil mi adaptación en el entorno laboral del CIADTI, donde he realizado el trabajo de grado en modalidad práctica empresarial. También expresar mi especial gratitud a los Ingenieros anteriormente mencionados Roger Legizamón y Andrés Villamizar quienes me compartieron parte de su conocimiento e hicieron posible el cumplimiento de este trabajo.

Tabla de contenido

RESUMEN

En este proyecto se ha realizado un procedimiento basado en varias metodologías ágiles de desarrollo de software, para implementar la aplicación móvil del sistema de "Peticiones, Quejas, Reclamos, Denuncias y Sugerencias (PQRDS) de la Universidad de Pamplona" con el objetivo de brindar un nuevo canal de comunicación para los usuarios de la Universidad; generando así el aumento de la calidad de los servicios prestados por parte de la Universidad a sus usuarios.

Para construir este procedimiento fue necesario realizar una revisión bibliográfica de las metodologías ágiles de desarrollo de software más populares, con el objetivo de encontrar la metodología de desarrollo que mejor se adaptase a la aplicación móvil a implementar y así adaptar los conceptos de la metodología al procedimiento; en este caso se adoptaron conceptos de varias metodologías con el objetivo de ofrecer un procedimiento sólido que se pudiera adecuar a las necesidades de la aplicación móvil.

Este trabajo se ha llevado a cabo en las instalaciones del CIADTI desarrollado bajo la modalidad de práctica empresarial, dirigida por el docente Edgar Albornoz y codirigida y supervisada por el coordinador técnico de desarrollo Wilfred Villalba.

Se pudo desarrollar en el tiempo estipulado la aplicación móvil gracias al procedimiento diseñado, el cual ha permitido realizar la implementación de manera organizada, ya que en todo el proceso enfocaba la atención del desarrollador en una tarea específica, viéndose reflejado esto en bajo tiempo de desarrollo y mayor calidad en cada una de sus funcionalidades.

Palabras Claves:

Metodología ágil, aplicación móvil, procedimiento de desarrollo móvil.

INTRODUCCIÓN

En estos últimos años se han implementado diferentes metodologías ágiles de desarrollo de software, pues permiten desarrollar productos de software con requisitos cambiantes y en un menor lapso de tiempo en comparación con las metodologías pesadas o tradicionales, y con la aparición y crecimiento de manera exponencial de los dispositivos móviles, el mercado demanda un gran número de aplicaciones para estos dispositivos en cortos periodos de tiempo, haciendo de las metodologías ágiles ideales para la implementación de estas aplicaciones.

A partir de los conceptos de diferentes metodologías ágiles se ha elaborado un procedimiento para implementar la aplicación móvil del sistema de "Peticiones, Quejas, Reclamos, Denuncias y Sugerencias (PQRDS) de la Universidad de Pamplona" con el objetivo de brindar un nuevo canal de comunicación para los usuarios de la Universidad; y por consecuencia de esto aumentar la calidad de los servicios prestados por parte de la Universidad a sus usuarios.

La construcción de este procedimiento se ha realizado después de una revisión bibliográfica de las metodologías ágiles de desarrollo de software más populares en la comunidad de desarrolladores, con el objetivo de encontrar la metodología de desarrollo que mejor se adaptase a la aplicación móvil a implementar y así adaptar los conceptos de la metodología al procedimiento, aunque gracias a las ventajas presentadas por las diferentes metodologías, se han tomado conceptos de varias de ellas, eso sí, complementándose cada uno de esos conceptos. Gracias al procedimiento producido, se ha podido implementar la aplicación PQRDS satisfactoriamente.

Este informe se encuentra estructurado por una serie de capítulos que describen la práctica llevada a cabo para el desarrollo de la aplicación móvil del sistema de PQRDS de la Universidad de Pamplona.

El capítulo 1 que se denomina Planteamiento del Problema contiene toda la información correspondiente al problema y justificación, proporcionando al lector la explicación necesaria para entender a la perfección la razón de ser de este proyecto, instaurando los objetivos correspondientes para llevar a cabo el cumplimiento de este.

El capítulo 2 que se denomina Marco Teórico y Estado del Arte, contiene información teórica que le da un soporte sólido a este proyecto, también contiene trabajos y/o proyectos realizados que apoyan el estudio del estado del arte.

El capítulo 3 que se denomina Metodología de Investigación, contiene el conjunto de pasos utilizados para desarrollar esta investigación, pasos que son de vital importancia ya que permiten desarrollar de manera organizada el proceso de investigación.

El capítulo 4 que se denomina Propuesta del Procedimiento y Resultados, contiene el producto final de este proyecto; este capítulo se divide en dos partes, el procedimiento para realizar la aplicación móvil del sistema de PQRDS de la Universidad de Pamplona y la aplicación móvil del sistema de PQRDS implementada a través del procedimiento.

CAPÍTULO 1. PLANTEAMIENTO DEL PROBLEMA

Este capítulo tiene toda la información correspondiente al problema y justificación, proporcionando al lector la explicación necesaria para entender a la perfección este proyecto, e instaurando los objetivos correspondientes para llevar a cabo el cumplimiento de este.

1.1 PROBLEMA

En los últimos años se ha notado un crecimiento de manera exponencial sobre el uso de los dispositivos y las aplicaciones móviles, ya sea motivado por el número de dispositivos, número de APPs disponibles o por la capacidad que tienen los SmartPhone. Esto significa que la tendencia por parte de los usuarios a tener toda la información necesaria en un móvil es cada vez mayor, en este sentido, el gobierno nacional ha diseñado estrategias para mantener la eficiencia, eficacia, visibilidad y publicidad en sus entidades.

La Universidad de Pamplona por ser una entidad pública debe adaptarse a los lineamientos del gobierno nacional, en este contexto, la Universidad para una eficiente gestión administrativa todo lo maneja a través de sistemas de información, donde el Centro de Investigación Aplicada y Desarrollo en Tecnologías de Información (CIADTI) es el encargado de resolver estas necesidades.

Uno de los sistemas de información de la Universidad es el aplicativo en línea de PQRDS, donde solo cuenta con acceso vía web (equipos de escritorio, etc.) y no se está cumpliendo con el acceso a través de dispositivos móviles, haciendo de esta una necesidad de desarrollo para el CIADTI.

1.2 JUSTIFICACIÓN

Dentro de los lineamientos del manual para la implementación de la estrategia de gobierno en línea para las entidades del orden nacional de la República de Colombia (MinTic, 2012), se establece que el sistema de contacto, peticiones, quejas, reclamos y denuncias puede tener acceso en los diferentes dispositivos móviles; en base a este lineamiento el CIADTI solicita la implementación del sistema de PQRDS para estos

dispositivos, por este motivo se pretende adaptar una metodología ágil en un procedimiento que ayude en el desarrollo de algunas aplicaciones móviles de los respectivos sistemas de información de la Universidad de Pamplona en el CIADTI, dentro de ellos la aplicación del sistema de PQRDS. Esta aplicación busca garantizar un nuevo canal de atención y comunicación de los usuarios de la Universidad de Pamplona a través de tecnologías móviles, para facilitar el seguimiento permanente.

Debido a que el manual para la implementación de la estrategia de gobierno en línea es de obligatorio cumplimiento para todas las entidades públicas y este sugiere la implementación del sistema de PQRDS en dispositivos móviles, se puede considerar relevante el desarrollo de este proyecto, que pretende aportar una solución para satisfacer una necesidad de la Universidad.

1.3 OBJETIVOS

1.3.1 OBJETIVO GENERAL

- ✓ Desarrollar un procedimiento basado en una metodología ágil para la implementación de la aplicación móvil del sistema de PQRDS de la Universidad de Pamplona en el CIADTI.

1.3.2 OBJETIVOS ESPECÍFICOS

- ✓ Realizar una revisión bibliográfica de las metodologías ágiles más utilizadas para el proceso de desarrollo de software.
- ✓ Identificar la metodología ágil que mejor se adapte al proceso de desarrollo de la aplicación móvil del sistema de PQRDS.
- ✓ Diseñar el procedimiento para el desarrollo de la aplicación móvil basado en la metodología.
- ✓ Desarrollar la aplicación móvil del sistema de PQRDS utilizando el procedimiento.
- ✓ Validar el procedimiento a través de la aplicación móvil del sistema de PQRDS en el CIADTI.

CAPÍTULO 2. MARCO TEÓRICO Y ESTADO DEL ARTE

Este capítulo contiene información teórica, que le da un soporte sólido a este proyecto, también contiene trabajos y/o proyectos realizados que apoyan el estudio del estado del arte.

2.1 MARCO TEÓRICO

Teniendo en cuenta que este proyecto tiene como objetivo desarrollar una aplicación para dispositivos móviles, aplicación que es desarrollada a través de un procedimiento basado en metodologías ágiles de desarrollo de software, a continuación se describen los conceptos involucrados en este proyecto.

2.1.1 PROCEDIMIENTO

Es una herramienta que tiene de forma consecutiva y organizada una serie de pasos para lograr una meta, en este caso para desarrollar la aplicación móvil de Peticiones, Quejas, Reclamos y Sugerencias de la Universidad de Pamplona.

2.1.2 METODOLOGÍAS DE DESARROLLO DE SOFTWARE

Para el desarrollo de software los grupos o las empresas desarrolladoras utilizan diferentes tipos de metodologías, dentro de ellas se encuentran las metodologías de desarrollo tradicional y la nueva tendencia que son las metodologías ágiles las cuales permiten terminar proyectos en un menor lapso de tiempo. A continuación se describen estas metodologías.

2.1.2.1 Metodología Tradicional

Una metodología de desarrollo de software se refiere al marco de trabajo que se utiliza para estructurar, planificar y controlar el proceso de desarrollo de un sistema de información. Una amplia variedad de esos marcos de trabajo han evolucionado a lo largo de los años, cada uno con sus propias fortalezas y debilidades reconocidas. Una metodología de desarrollo de software no es necesariamente adecuado para su uso en todos los proyectos. Cada una de las metodologías disponibles se adecua para

determinados tipos de proyectos, basado en diversas consideraciones técnicas, organizativas y de equipo (Services Office of information, Selecting a development approach. Revalidated: 2008, p.1).

Las metodologías de desarrollo tradicionales tienen como características principales algúnos de los siguientes conceptos:

Modelo en Cascada:

El modelo de cascada tiene sus orígenes en la década de 1970, y se define como una secuencia de actividades bien planificadas y estructuradas (ver *Figura 2.1*). El proceso distingue claramente las fases de especificación de las de desarrollo y éstas, a su vez, de las de testing. Es, seguramente, la metodología más extendida y utilizada. Este modelo se basa fuertemente en que cada detalle de los requisitos se conoce de antemano, previo de comenzar la fase de codificación o desarrollo, y asume, además, que no existirán cambios significativos en los mismos a lo largo del ciclo de vida del desarrollo (Joskowicz, 2008, p. 6).

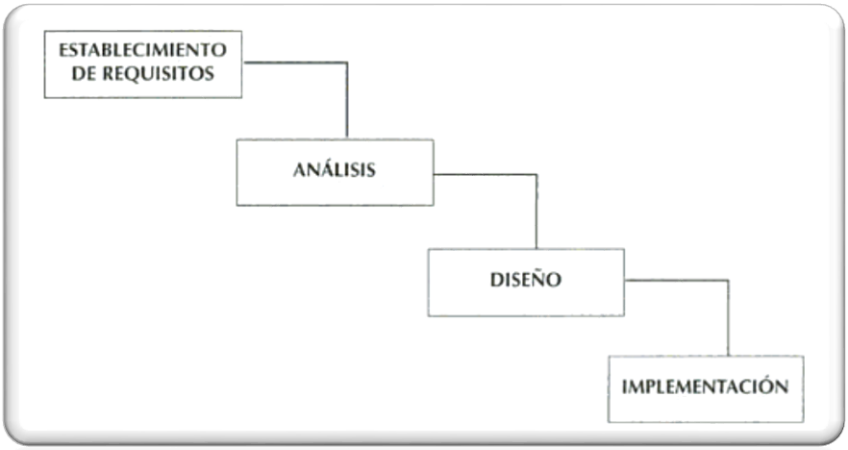


Figura 2.1. Modelo Cascada.
Cortés Morales, (s.f). Introducción al Análisis de Sistemas y la Ingeniería de Software.

Modelo Incremental:

El modelo incremental consiste en un desarrollo inicial de la arquitectura completa del sistema, seguido de sucesivos incrementos funcionales (ver *Figura 2.2*). Cada incremento tiene su propio ciclo de vida y se basa en el anterior, sin cambiar su funcionalidad ni sus interfaces. Una vez entregado un incremento, no se realizan cambios sobre el mismo, sino únicamente corrección de errores. Dado que la arquitectura completa se desarrolla en la etapa inicial, es necesario, al igual que en el modelo en cascada, conocer los requerimientos completos al comienzo del desarrollo. Respecto al modelo en cascada, el incremental tiene la ventaja de entregar una funcionalidad inicial en menor tiempo (Joskowicz, 2008, p. 6).

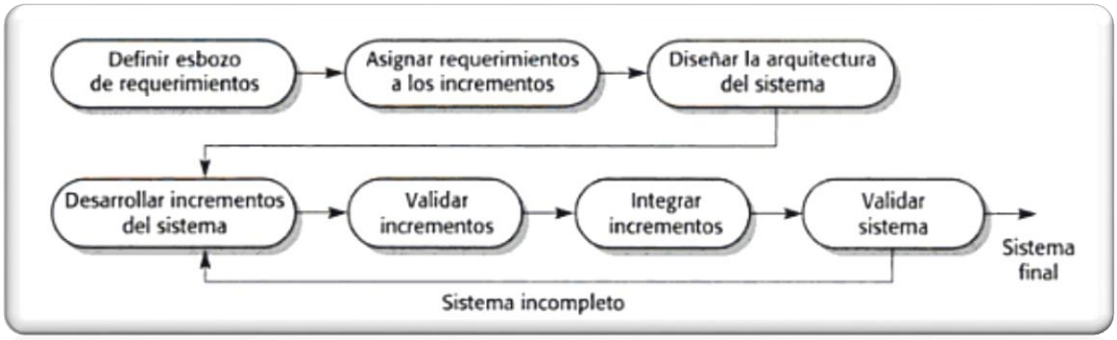


Figura 2.2. Modelo Incremental.
Sommerville, (2005). Ingeniería del Software.

Modelo Evolutivo:

El modelo evolutivo es, en cierta forma, similar al incremental, pero admite que la especificación no esté completamente determinada al comienzo del ciclo de vida (ver *Figura 2.3*). Los requerimientos que estén suficientemente detallados al comienzo darán lugar a una entrega inicial, mientras que los siguientes incrementos serán cambios progresivos que implementen "deltas" de especificación de requerimientos. El modelo admite que, si la especificación no es suficientemente clara al principio, puede desarrollarse un prototipo experimental, que tiene como función validar o identificar los requisitos del sistema (Joskowicz, 2008, p. 7).

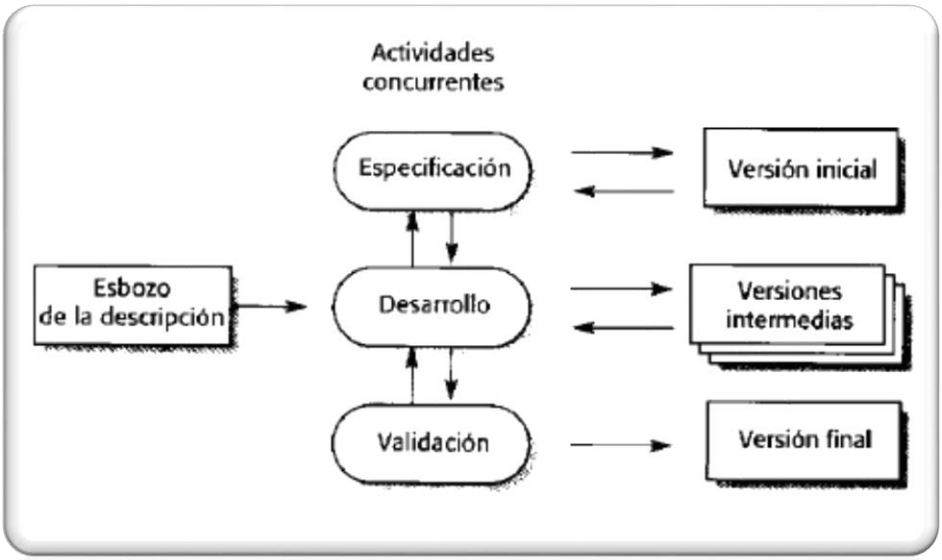


Figura 2.3. Modelo Evolutivo.
Sommerville, (2005). Ingeniería del Software.

Modelo Espiral:

El modelo de espiral, introducido por Barry Bohem a fines de la década de 1980, intenta combinar las ventajas del modelo en cascada con el modelo evolutivo (ver *Figura 2.4*). El modelo enfatiza el estudio de los riesgos del proyecto, como por ejemplo las especificaciones incompletas. Se prevé, en este modelo, varios ciclos o "vueltas de espiral", cada uno de ellos con cuatro etapas: Definición de objetivos, Evaluación y reducción del riesgo, Desarrollo y validación y Planificación del siguiente ciclo. En este modelo, una actividad comienza solo cuando se entienden los objetivos y riesgos involucrados. El desarrollo se incrementa en cada etapa, generando una solución completa (Joskowicz, 2008, p. 7).

Figura 2.4. Modelo Espiral.
Cortés Morales, (s.f). Introducción al Análisis de Sistemas y la Ingeniería de Software.

2.1.2.2 Metodología Ágil

"Es el conjunto de buenos valores y buenas prácticas para el desarrollo de proyectos de software" (Gamboa Manzaba según su cita del Manifiesto Ágil, 2011).

Las metodologías ágiles surgen como contraste de las metodologías tradicionales y consisten en una serie de pasos que permiten a un grupo de desarrolladores llevar a cabo eficientemente (tiempo, flexibilidad en cuanto a modificaciones se refiere, costos, etc.) los procesos de desarrollo de software. Los métodos ágiles enfatizan las comunicaciones cara a cara y deben cumplir con los principios establecidos en el manifiesto ágil.

La siguiente descripción es tomada del manifiesto ágil (Beck, y otros, 2001):

Manifiesto por el Desarrollo Ágil de Software

Es un escrito donde se establecen los valores que los métodos de desarrollo de software deben seguir para considerarse métodos ágiles. Según el manifiesto se deben valorar:

1. **Individuos e interacciones** sobre procesos y herramientas.
2. **Software funcionando** sobre documentación extensiva.
3. **Colaboración con el cliente** sobre negociación contractual.
4. **Respuesta ante el cambio** sobre seguir un plan.

A partir de estos valores surgieron los siguientes principios:

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.

5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7. El software funcionando es la medida principal de progreso.
8. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

A continuación se redactan las metodologías ágiles de desarrollo de software competentes a este proyecto:

➢ **Extreme Programming (XP):**

Es una metodología ágil 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 programadores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones y coraje para enfrentar los cambios (ver *Figura 2.5*). XP es adecuada para proyectos con requisitos imprecisos y cambiantes (Orjuela Duarte & Rojas según citan de Beck, 2008, p. 162).

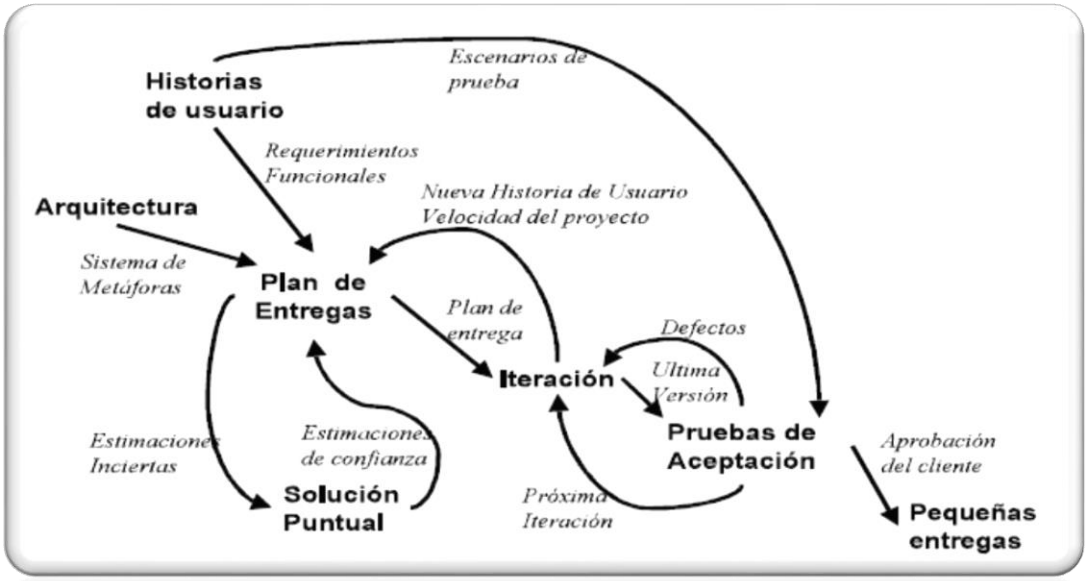


Figura 2.5. Proceso XP.
Orjuela Duarte & Rojas, (2008). Las Metodologías de Desarrollo Ágil como una Oportunidad para la Ingeniería del Software Educativo.

A continuación se describen las principales características de la metodología XP que están compuestas básicamente por fases, reglas y valores:

***Fase de Exploración*:**

Es la fase en la que se define el alcance general del proyecto. En esta fase, el cliente define lo que necesita mediante la redacción de sencillas "historias de usuarios". Los programadores estiman los tiempos de desarrollo en base a esta información. Debe quedar claro que las estimaciones realizadas en esta fase son primarias (ya que estarán basadas en datos de muy alto nivel), y podrían variar cuando se analicen más en detalle en cada iteración. Esta fase dura típicamente un par de semanas, y el resultado es una visión general del sistema, y un plazo total estimado (Joskowicz, 2008, p. 8).

***Fase de Planificación*:**

La planificación es una fase corta, en la que el cliente, los gerentes y el grupo de desarrolladores acuerdan el orden en que deberán implementarse las historias de usuario, y, asociadas a éstas, las entregas. Típicamente esta fase consiste en una o varias reuniones grupales de planificación. El resultado de esta fase es un Plan de Entregas, o "Release Plan" (Joskowicz, 2008, p. 9).

***Fase de Iteraciones*:**

Esta es la fase principal en el ciclo de desarrollo de XP. Las funcionalidades son desarrolladas en esta fase, generando al final de cada una un entregable funcional que implementa las historias de usuario asignadas a la iteración. Como las historias de usuario no tienen suficiente detalle como para permitir su análisis y desarrollo, al principio de cada iteración se realizan las tareas necesarias de análisis, recabando con el cliente todos los datos que sean necesarios. El cliente, por lo tanto, también debe participar activamente durante esta fase del ciclo. Las iteraciones son también utilizadas para medir el progreso del proyecto. Una iteración terminada sin errores es una medida clara de avance (Joskowicz, 2008, p. 9).

***Fase de Puesta en Producción*:**

Si bien al final de cada iteración se entregan módulos funcionales y sin errores, puede ser deseable por parte del cliente no poner el sistema en producción hasta tanto no se tenga la funcionalidad completa. En esta fase no se realizan más desarrollos funcionales, pero pueden ser necesarias tareas de ajuste (Joskowicz, 2008, p. 9).

***Reglas y Prácticas*:**

La metodología XP tiene un conjunto importante de reglas y prácticas para realizar la planificación, el diseño, el desarrollo y las pruebas.

Planificación

La metodología XP plantea la planificación como un dialogo continuo entre las partes involucradas en el proyecto, incluyendo al cliente, a los programadores y a los coordinadores o gerentes. El proyecto comienza recopilando "Historias de usuarios", las que sustituyen a los tradicionales "casos de uso". Una vez obtenidas las "historias de usuarios", los programadores evalúan rápidamente el tiempo de desarrollo de cada una. Si alguna de ellas tiene "riesgos" que no permiten establecer con certeza la complejidad del desarrollo, se realizan pequeños programas de prueba ("spikes"), para reducir estos riesgos. Una vez realizadas estas estimaciones, se organiza una reunión de planificación, con los diversos actores del proyecto (cliente, desarrolladores, gerentes), a los efectos de establecer un plan o cronograma de entregas ("Release Plan") en los que todos estén de acuerdo. Una vez acordado este cronograma, comienza una fase de iteraciones, en dónde en cada una de ellas se desarrolla, prueba e instala unas pocas "historias de usuarios" (Joskowicz, 2008, p. 9).

Como cita Joskowicz (2008) "según Martín Fowler (uno de los firmantes del "Agile Manifesto"), los planes en XP se diferencian de las metodologías tradicionales en tres aspectos" (p. 10):

- Simplicidad del plan. No se espera que un plan requiera de un "gurú" con complicados sistemas de gerenciamiento de proyectos.
- Los planes son realizados por las mismas personas que realizarán el trabajo.
- Los planes no son predicciones del futuro, sino simplemente la mejor estimación de cómo saldrán las cosas. Los planes son útiles, pero necesitan ser cambiados cuando las circunstancias lo requieren. De otra manera, se termina en situaciones en las que el plan y la realidad no coinciden, y en estos casos, el plan es totalmente inútil.

Los conceptos básicos de esta planificación son los siguientes:

***Historias de Usuario*:** Las "Historias de usuarios" ("User stories") sustituyen a los documentos de especificación funcional, y a los "casos de uso". Estas "historias" son escritas por el cliente, en su propio lenguaje, como descripciones cortas de lo que el sistema debe realizar. La diferencia más importante entre estas historias y los tradicionales documentos de especificación funcional se encuentra en el nivel de detalle requerido. Las historias de usuario deben tener el detalle mínimo como para que los programadores puedan realizar una estimación poco riesgosa del tiempo que llevará su desarrollo. Cuando llegue el momento de la implementación, los desarrolladores dialogarán directamente con el cliente para obtener todos los detalles necesarios. Las historias de usuarios deben poder ser programadas en un tiempo entre una y tres semanas. Si la estimación es superior a tres semanas, debe ser dividida en dos o más historias. Si es menos de una semana, se debe combinar con otra historia (Joskowicz, 2008, p. 10).

***Plan de Entregas*:** El cronograma de entregas establece qué historias de usuario serán agrupadas para conformar una entrega, y el orden de las mismas. Este cronograma será el resultado de una reunión entre todos los actores del proyecto (cliente, desarrolladores, gerentes, etc.). XP denomina a esta reunión "Juego de planeamiento" ("Planning game"), pero puede denominarse de la manera que sea más apropiada al tipo de empresa y cliente (por ejemplo, Reunión de planeamiento, "Planning meeting" o "Planning workshop"). Típicamente el cliente ordenará y agrupará según sus prioridades las historias de usuario; el cronograma de entregas se realiza en base a las estimaciones de tiempos de desarrollo realizadas por los desarrolladores. Luego de algunas iteraciones es recomendable realizar nuevamente una reunión con los actores del proyecto, para evaluar nuevamente el plan de entregas y ajustarlo si es necesario (Joskowicz, 2008, p. 10).

***Plan de Iteraciones*:** Las historias de usuarios seleccionadas para cada entrega son desarrolladas y probadas en un ciclo de iteración, de acuerdo al orden preestablecido. Al comienzo de cada ciclo, se realiza una reunión de planificación de la iteración. Cada historia de usuario se traduce en tareas específicas de programación; asimismo, para cada historia de usuario se establecen las pruebas de aceptación. Estas pruebas se realizan al final del ciclo en el que se desarrollan, pero también al final de cada uno de los ciclos siguientes, para verificar que subsiguientes iteraciones no han afectado a las

anteriores. Las pruebas de aceptación que hayan fallado en el ciclo anterior son analizadas para evaluar su corrección, así como para prever que no vuelvan a ocurrir (Joskowicz, 2008, p. 11).

***Reuniones Diarias de Seguimiento*:** El objetivo de tener reuniones diarias es mantener la comunicación entre el equipo, y compartir problemas y soluciones. En la mayoría de estas reuniones, gran parte de los participantes simplemente escuchan, sin tener mucho que aportar. Para no quitar tiempo innecesario del equipo, se sugiere realizar estas reuniones en círculo y de pie (Joskowicz, 2008, p. 11).

Diseño

La metodología XP hace especial énfasis en los diseños simples y claros. Los conceptos más importantes de diseño en esta metodología son los siguientes:

***Simplicidad*:** Un diseño simple se implementa más rápidamente que uno complejo. Por ello XP propone implementar el diseño más simple posible que funcione. Se sugiere nunca adelantar la implementación de funcionalidades que no correspondan a la iteración en la que se esté trabajando (Joskowicz, 2008, p. 11).

***Soluciones*:** Cuando aparecen problemas técnicos, o cuando es difícil de estimar el tiempo para implementar una historia de usuario, pueden utilizarse pequeños programas de prueba (llamados "spike"), para explorar diferentes soluciones. Estos programas son únicamente para probar o evaluar una solución, y suelen ser desechados luego de su evaluación (Joskowicz, 2008, p. 11).

***Recodificación*:** La recodificación ("refactoring") consiste en escribir nuevamente parte del código de un programa, sin cambiar su funcionalidad, a los efectos de hacerlo más simple, conciso y/o entendible. Muchas veces, al terminar de escribir un código de programa, pensamos que, si lo comenzáramos de nuevo, lo hubiéramos hecho en forma diferente, más clara y eficientemente. Sin embargo, como ya está pronto y "funciona", rara vez es reescrito. Las metodologías de XP sugieren recodificar cada vez que sea necesario. Si bien, puede parecer una pérdida de tiempo innecesaria en el plazo inmediato, los resultados de ésta práctica tienen sus frutos en las siguientes iteraciones, cuando sea necesario ampliar o cambiar la funcionalidad. La filosofía que

se persigue es, como ya se mencionó, tratar de mantener el código más simple posible que implemente la funcionalidad deseada (Joskowicz, 2008, p. 12).

***Metáforas*:** Una "metáfora" es algo que todos entienden, sin necesidad de mayores explicaciones. La metodología XP sugiere utilizar este concepto como una manera sencilla de explicar el propósito del proyecto, y guiar la estructura y arquitectura del mismo. Por ejemplo, puede ser una guía para la nomenclatura de los métodos y las clases utilizadas en el diseño del código. Tener nombres claros, que no requieran de mayores explicaciones, redunda en un ahorro de tiempo. Es muy importante que el cliente y el grupo de desarrolladores estén de acuerdo y compartan esta "metáfora", para que puedan dialogar en un "mismo idioma". Una buena metáfora debe ser fácil de comprender para el cliente y a su vez debe tener suficiente contenido como para que sirva de guía a la arquitectura del proyecto. Sin embargo, ésta práctica resulta, muchas veces, difícil de realizar (Joskowicz, 2008, p. 12).

Desarrollo del Código

***Disponibilidad del Cliente*:** Uno de los requerimientos de XP es tener al cliente disponible durante todo el proyecto. No solamente como apoyo a los desarrolladores, sino formando parte del grupo. El involucramiento del cliente es fundamental para que pueda desarrollarse un proyecto con la metodología XP (Joskowicz, 2008, p. 12).

Al comienzo del proyecto, el cliente debe proporcionar las historias de usuarios. Pero, dado que estas historias son expresamente cortas y de "alto nivel", no contienen los detalles necesarios para realizar el desarrollo del código. Estos detalles deben ser proporcionados por el cliente, y discutidos con los desarrolladores, durante la etapa de desarrollo. No se requieren de largos documentos de especificaciones, sino que los detalles son proporcionados por el cliente, en el momento adecuado, "cara a cara" a los desarrolladores. Si bien esto parece demandar del cliente recursos por un tiempo prolongado, debe tenerse en cuenta que en otras metodologías este tiempo es insumido por el cliente en realizar los documentos detallados de especificación. Adicionalmente, al estar el cliente en todo el proceso, puede prevenir a tiempo de situaciones no deseables, o de funcionamientos que no eran los que en realidad se deseaban. En otras metodologías, estas situaciones son detectadas en forma muy tardía del ciclo de desarrollo, y su corrección puede llegar a ser muy complicada (Joskowicz, 2008, p. 12).

***Uso de Estándares*:** "Si bien esto no es una idea nueva, XP promueve la programación basada en estándares, de manera que sea fácilmente entendible por todo el equipo, y que facilite la recodificación" (Joskowicz, 2008, p. 13).

***Programación Dirigida por las Pruebas*:** En las metodologías tradicionales, la fase de pruebas, incluyendo la definición de los tests, es usualmente realizada sobre el final del proyecto, o sobre el final del desarrollo de cada módulo. La metodología XP propone un modelo inverso, en el que, lo primero que se escribe son los test que el sistema debe pasar. Luego, el desarrollo debe ser el mínimo necesario para pasar las pruebas previamente definidas. Las pruebas a las que se refiere esta práctica, son las pruebas unitarias, realizados por los desarrolladores. La definición de estos test al comienzo, condiciona o "dirige" el desarrollo (Joskowicz, 2008, p. 13).

***Programación en Parejas*:** XP propone que se desarrolle en pares de programadores, ambos trabajando juntos en un mismo ordenador. Si bien parece que ésta práctica duplica el tiempo asignado al proyecto (y por ende, los costos en recursos humanos), al trabajar en pares se minimizan los errores y se logran mejores diseños, compensando la inversión en horas. El producto obtenido es por lo general de mejor calidad que cuando el desarrollo se realiza por programadores individuales (Joskowicz, 2008, p. 13).

En un estudio realizado por Cockburn y Williams, se concluye que la programación en pares tiene un sobre costo aproximado de 15%, y no de un 100% como se puede pensar a priori. Este sobre costo es rápidamente pagado por la mejor calidad obtenida en el producto final (Joskowicz, 2008, p. 13).

Adicionalmente, como redacta Joskowicz (2008) "la programación en pares tiene las siguientes ventajas" (p. 13):

- La mayoría de los errores se descubren en el momento en que se codifican, ya que el código es permanentemente revisado por dos personas.
- La cantidad de defectos encontrados en las pruebas es estadísticamente menor.
- Los diseños son mejores y el código más corto.
- El equipo resuelve problemas en forma más rápida.

- Las personas aprenden significativamente más, acerca del sistema y acerca de desarrollo de software.
- El proyecto termina con más personas que conocen los detallas de cada parte del código.
- Las personas aprenden a trabajar juntas, generando mejor dinámica de grupo y haciendo que la información fluya rápidamente.
- Las personas disfrutan más de su trabajo.

***Integraciones Permanentes*:** Todos los desarrolladores necesitan trabajar siempre con la "última versión". Realizar cambios o mejoras sobre versiones antiguas causan graves problemas, y retrasan al proyecto. Es por eso que XP promueve publicar lo antes posible las nuevas versiones, aunque no sean las últimas, siempre que estén libres de errores. Idealmente, todos los días deben existir nuevas versiones publicadas. Para evitar errores, solo una pareja de desarrolladores puede integrar su código a la vez (Joskowicz, 2008, p. 14).

***Propiedad Colectiva del Código*:** "En un proyecto XP, todo el equipo puede contribuir con nuevas ideas que apliquen a cualquier parte del proyecto. Asimismo, cualquier pareja de programadores puede cambiar el código que sea necesario para corregir problemas, agregar funciones o recodificar" (Joskowicz, 2008, p.14).

En otras metodologías, este concepto puede parecer extraño. Muchas veces se asume que, si hay algo de propiedad colectiva, la responsabilidad también es colectiva. Y que "todos sean responsables", muchas veces significa que "nadie es responsable". Ward Cunningham explica en una entrevista con Bill Veners, que este razonamiento no es correcto cuando se trabaja con la metodología de XP. En este caso, quienes encuentran un problema, o necesitan desarrollar una nueva función, pueden resolverlo directamente, sin necesidad de "negociar" con el "dueño" o autor del módulo (ya que, de hecho, este concepto no existe en XP). Muchas veces, explica Cunningham, una solución pasa por la recodificación de varios módulos, que atraviesan de forma horizontal una determinada jerarquía vertical. Si es necesario dialogar y convencer al encargado de cada módulo, posiblemente la solución no se pueda implementar, por lo menos en tiempos razonables. En XP, se promueve la recodificación, en aras de generar códigos más simples y adaptados a las realidades cambiantes. Cualquier pareja de

programadores puede tomar la responsabilidad de este cambio. Los testeos permanentes deberían asegurar que los cambios realizados cumplen con lo requerido, y además, que no afectan al resto de las funcionalidades (Joskowicz, 2008, p.14).

***Ritmo Sostenido*:** La metodología XP indica que debe llevarse un ritmo sostenido de trabajo. Anteriormente, ésta práctica se denominaba "Semana de 40 horas". Sin embargo, lo importante no es si se trabajan, 35, 40 o 42 horas por semana. El concepto que se desea establecer con esta práctica es el de planificar el trabajo de manera de mantener un ritmo constante y razonable, sin sobrecargar al equipo (Joskowicz, 2008, p.14).

Cuando un proyecto se retrasa, trabajar tiempo extra puede ser más perjudicial que beneficioso. El trabajo extra desmotiva inmediatamente al grupo e impacta en la calidad del producto. En la medida de lo posible, se debería renegociar el plan de entregas ("Release Plan"), realizando una nueva reunión de planificación con el cliente, los desarrolladores y los gerentes. Adicionalmente, agregar más desarrolladores en proyectos ya avanzados no siempre resuelve el problema (Joskowicz, 2008, p.14).

Pruebas

***Pruebas Unitarias*:** Las pruebas unitarias son una de las piedras angulares de XP. Todos los módulos deben de pasar las pruebas unitarias antes de ser liberados o publicados. Por otra parte, como se mencionó anteriormente, las pruebas deben ser definidas antes de realizar el código ("Test-driven programming"). Que todo código liberado pase correctamente las pruebas unitarias es lo que habilita que funcione la propiedad colectiva del código. En este sentido, el sistema y el conjunto de pruebas debe ser guardado junto con el código, para que pueda ser utilizado por otros desarrolladores, en caso de tener que corregir, cambiar o recodificar parte del mismo (Joskowicz, 2008, p. 15).

***Detección y Corrección de Errores*:** "Cuando se encuentra un error ("bug"), éste debe ser corregido inmediatamente, y se deben tener precauciones para que errores similares no vuelvan a ocurrir. Asimismo, se generan nuevas pruebas para verificar que el error haya sido resuelto" (Joskowicz, 2008, p. 15).

***Pruebas de Aceptación*:** Las pruebas de aceptación son creadas en base a las historias de usuarios, en cada ciclo de la iteración del desarrollo. El cliente debe especificar uno o diversos escenarios para comprobar que una historia de usuario ha sido correctamente implementada. Las pruebas de aceptación son consideradas como "pruebas de caja negra" ("black box system tests"). Los clientes son responsables de verificar que los resultados de estas pruebas sean correctos. Asimismo, en caso de que fallen varias pruebas, deben indicar el orden de prioridad de resolución. Una historia de usuario no se puede considerar terminada hasta tanto pase correctamente todas las pruebas de aceptación. Dado que la responsabilidad es grupal, es recomendable publicar los resultados de las pruebas de aceptación, de manera que todo el equipo esté al tanto de esta información (Joskowicz, 2008, p. 15).

Valores en XP

Los siguientes valores deben estar presente en todo momento en el equipo de desarrollo para que el proyecto termine exitosamente:

***Comunicación*:** Muchos de los problemas que existen en proyectos de software (así como en muchos otros ámbitos) se deben a problemas de comunicación entre las personas. La comunicación permanente es fundamental en XP. Dado que la documentación es escasa, el diálogo frontal, cara a cara, entre desarrolladores, gerentes y el cliente es el medio básico de comunicación. Una buena comunicación tiene que estar presente durante todo el proyecto (Joskowicz, 2008, p. 16).

***Simplicidad*:** XP, como metodología ágil, apuesta a la sencillez, en su máxima expresión. Sencillez en el diseño, en el código, en los procesos, etc. La sencillez es esencial para que todos puedan entender el código, y se trata de mejorar mediante recodificaciones continuas (Joskowicz, 2008, p. 16).

***Retroalimentación*:** La retroalimentación debe funcionar en forma permanente. El cliente debe brindar retroalimentación de las funciones desarrolladas, de manera de poder tomar sus comentarios para la próxima iteración, y para comprender, cada vez más, sus necesidades. Los resultados de las pruebas unitarias son también una retroalimentación permanente que tienen los desarrolladores acerca de la calidad de su trabajo (Joskowicz, 2008, p. 16).

***Coraje*:** Cuando se encuentran problemas serios en el diseño, o en cualquier otro aspecto, se debe tener el coraje suficiente como para encarar su solución, sin importar que tan difícil sea. Si es necesario cambiar completamente parte del código, hay que hacerlo, sin importar cuanto tiempo se ha invertido previamente en el mismo (Joskowicz, 2008, p. 16).

➢ **Scrum:**

Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos años. Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas sprint, con una duración de 30 días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo del proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración (ver *Figura 2.6*) (Orjuela Duarte & Rojas según citan de Gutierrez, 2008, p. 166).

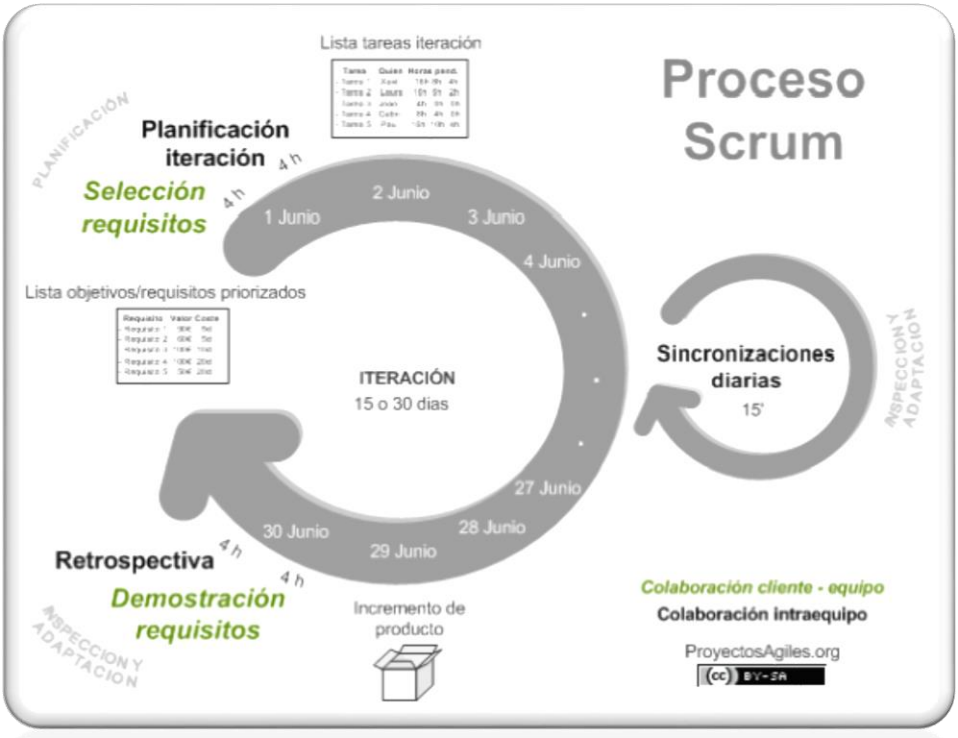


Figura 2.6. Proceso Scrum.
Monge Fallas, (2012). Metodologías ágiles aplicadas a la Administración de Proyectos de Desarrollo de Software.

"Tres pilares soportan toda la implementación del control de procesos empírico: transparencia, inspección y adaptación" (Schwaber & Sutherland, 2013, p. 4).

Transparencia

Los aspectos significativos del proceso deben ser visibles para aquellos que son responsables del resultado. La transparencia requiere que dichos aspectos sean definidos por un estándar común, de tal modo que los observadores compartan un entendimiento común de lo que se está viendo (Schwaber & Sutherland, 2013, p. 5).

Inspección

Los usuarios de Scrum deben inspeccionar frecuentemente los artefactos de Scrum y el progreso hacia un objetivo, para detectar variaciones. Su inspección no debe ser tan frecuente como para que interfiera en el trabajo. Las inspecciones son más beneficiosas cuando se realizan de forma diligente por inspectores expertos, en el mismo lugar de trabajo (Schwaber & Sutherland, 2013, p. 5).

Adaptación

Si un inspector determina que uno o más aspectos de un proceso se desvían de límites aceptables, y que el producto resultante no será aceptable, el proceso o el material que está siendo procesado deben ser ajustados. Dicho ajuste debe realizarse cuanto antes para minimizar desviaciones mayores (Schwaber & Sutherland, 2013, p. 5).

Como redactan Schwaber & Sutherland (2013), "Scrum prescribe cuatro eventos formales, contenidos dentro del Sprint, para la inspección y adaptación" (p. 5):

- Reunión de Planificación del Sprint (Sprint Planning Meeting)
- Scrum Diario (Daily Scrum)
- Revisión del Sprint (Sprint Review)
- Retrospectiva del Sprint (Sprint Retrospective)

La estructura de Scrum la conforma el Equipo Scrum, los Eventos y los Artefactos que son descritos a continuación:

El Equipo Scrum (Scrum Team)

El Equipo Scrum consiste en un Dueño de Producto (Product Owner), el Equipo de Desarrollo (Development Team) y un Scrum Master. Los Equipos Scrum son autoorganizados y multifuncionales. Los equipos autoorganizados eligen la mejor forma de llevar a cabo su trabajo y no son dirigidos por personas externas al equipo. Los equipos multifuncionales tienen todas las competencias necesarias para llevar a cabo el trabajo sin depender de otras personas que no son parte del equipo. El modelo de equipo en Scrum está diseñado para optimizar la flexibilidad, la creatividad y la productividad (Schwaber & Sutherland, 2013, p. 5).

"Los Equipos Scrum entregan productos de forma iterativa e incremental, maximizando las oportunidades de obtener retroalimentación. Las entregas incrementales de producto "Terminado" aseguran que siempre estará disponible una versión potencialmente útil y funcional del producto" (Schwaber & Sutherland, 2013, p. 6).

El Dueño de Producto (Product Owner)

"El Dueño de Producto es el responsable de maximizar el valor del producto y del trabajo del Equipo de Desarrollo. El cómo se lleva a cabo esto podría variar ampliamente entre distintas organizaciones, Equipos Scrum e individuos" (Schwaber & Sutherland, 2013, p. 6).

Schwaber & Sutherland (2013) señalan que "el Dueño de Producto es la única persona responsable de gestionar la Lista del Producto (Product Backlog)" (p. 6). La gestión de la Lista del Producto incluye:

- Expresar claramente los elementos de la Lista del Producto;
- Ordenar los elementos en la Lista del Producto para alcanzar los objetivos y misiones de la mejor manera posible;
- Optimizar el valor del trabajo desempeñado por el Equipo de Desarrollo;
- Asegurar que la Lista del Producto es visible, transparente y clara para todos, y que muestra aquello en lo que el equipo trabajará a continuación; y,

- Asegurar que el Equipo de Desarrollo entiende los elementos de la Lista del Producto al nivel necesario.

"El Dueño de Producto podría hacer el trabajo anterior, o delegarlo en el Equipo de Desarrollo. Sin embargo, en ambos casos el Dueño de Producto sigue siendo el responsable de dicho trabajo" (Schwaber & Sutherland, 2013, p. 6).

El Dueño de Producto es una única persona, no un comité. El Dueño de Producto podría representar los deseos de un comité en la Lista del Producto, pero aquellos que quieran cambiar la prioridad de un elemento de la Lista deben hacerlo a través del Dueño de Producto (Schwaber & Sutherland, 2013, p. 6).

El Equipo de Desarrollo (Development Team)

El Equipo de Desarrollo consiste en los profesionales que desempeñan el trabajo de entregar un Incremento de producto "Terminado", que potencialmente se pueda poner en producción, al final de cada Sprint. Solo los miembros del Equipo de Desarrollo participan en la creación del Incremento. Los Equipos de Desarrollo son estructurados y empoderados por la organización para organizar y gestionar su propio trabajo. La sinergia resultante optimiza la eficiencia y efectividad del Equipo de Desarrollo (Schwaber & Sutherland, 2013, p. 7).

Según Schwaber & Sutherland (2013) los equipos de desarrollo tienen las siguientes características:

- Son autoorganizados. Nadie (ni siquiera el Scrum Master) indica al Equipo de Desarrollo cómo convertir elementos de la Lista del Producto en Incrementos de funcionalidad potencialmente desplegables;
- Los Equipos de Desarrollo son multifuncionales, contando como equipo con todas las habilidades necesarias para crear un Incremento de producto;
- Scrum no reconoce títulos para los miembros de un Equipo de Desarrollo, todos son Desarrolladores, independientemente del trabajo que realice cada persona; no hay excepciones a esta regla;

- Scrum no reconoce sub-equipos en los equipos de desarrollo, no importan los dominios particulares que requieran ser tenidos en cuenta, como pruebas o análisis de negocio; no hay excepciones a esta regla; y,
- Los Miembros individuales del Equipo de Desarrollo pueden tener habilidades especializadas y áreas en las que estén más enfocados, pero la responsabilidad recae en el Equipo de Desarrollo como un todo.

Tamaño del Equipo de Desarrollo

El tamaño óptimo del Equipo de Desarrollo es lo suficientemente pequeño como para permanecer ágil y lo suficientemente grande como para completar una cantidad de trabajo significativa. Tener menos de tres miembros en el Equipo de Desarrollo reduce la interacción y resulta en ganancias de productividad más pequeñas. Los Equipos de Desarrollo más pequeños podrían encontrar limitaciones en cuanto a las habilidades necesarias durante un Sprint, haciendo que el Equipo de Desarrollo no pudiese entregar un Incremento que potencialmente se pueda poner en producción. Tener más de nueve miembros en el equipo requiere demasiada coordinación. Los Equipos de Desarrollo grandes generan demasiada complejidad como para que pueda gestionarse mediante un proceso empírico. Los roles de Dueño de Producto y Scrum Master no cuentan en el cálculo del tamaño del equipo a menos que también estén contribuyendo a trabajar en la Lista de Pendientes de Sprint (Sprint Backlog). (Schwaber & Sutherland, 2013, p. 7)

El Scrum Master

El Scrum Master es el responsable de asegurar que Scrum es entendido y adoptado. Los Scrum Masters hacen esto asegurándose de que el Equipo Scrum trabaja ajustándose a la teoría, prácticas y reglas de Scrum; el Scrum Master es un líder que está al servicio del Equipo Scrum. Este líder ayuda a las personas externas al Equipo Scrum a entender qué interacciones con el Equipo Scrum pueden ser de ayuda y cuáles no. El Scrum Master ayuda a todos a modificar estas interacciones para maximizar el valor creado por el Equipo Scrum. (Schwaber & Sutherland, 2013, p. 8)

El Servicio del Scrum Master al Dueño de Producto

Según Schwaber & Sutherland (2013), el Scrum Master da servicio al Dueño de Producto de varias formas, incluyendo:

- Encontrar técnicas para gestionar la Lista de Producto de manera efectiva;
- Ayudar al Equipo Scrum a entender la necesidad de contar con elementos de Lista de Producto claros y concisos;
- Entender la planificación del producto en un entorno empírico;
- Asegurar que el Dueño de Producto conozca cómo ordenar la Lista de Producto para maximizar el valor;
- Entender y practicar la agilidad; y,
- Facilitar los eventos de Scrum según se requiera o necesite.

El Servicio del Scrum Master al Equipo de Desarrollo

Según Schwaber & Sutherland (2013), el Scrum Master da servicio al Equipo de Desarrollo de varias formas, incluyendo:

- Guiar al Equipo de Desarrollo en ser autoorganizado y multifuncional;
- Ayudar al Equipo de Desarrollo a crear productos de alto valor;
- Eliminar impedimentos para el progreso del Equipo de Desarrollo;
- Facilitar los eventos de Scrum según se requiera o necesite; y,
- Guiar al Equipo de Desarrollo en el entorno de organizaciones en las que Scrum aún no ha sido adoptado y entendido por completo.

El Servicio del Scrum Master a la Organización

Según Schwaber & Sutherland (2013), el Scrum Master da servicio a la organización de varias formas, incluyendo:

- Liderar y guiar a la organización en la adopción de Scrum;
- Planificar las implementaciones de Scrum en la organización;
- Ayudar a los empleados e interesados a entender y llevar a cabo Scrum y el desarrollo empírico de producto;
- Motivar cambios que incrementen la productividad del Equipo Scrum; y,

- Trabajar con otros Scrum Masters para incrementar la efectividad de la aplicación de Scrum en la organización.

Eventos de Scrum

En Scrum existen eventos predefinidos con el fin de crear regularidad y minimizar la necesidad de reuniones no definidas en Scrum. Todos los eventos son bloques de tiempo (time-boxes), de tal modo que todos tienen una duración máxima. Una vez que comienza un Sprint, su duración es fija y no puede acortarse o alargarse. Los demás eventos pueden terminar siempre que se alcance el objetivo del evento, asegurando que se emplee una cantidad apropiada de tiempo sin permitir desperdicio en el proceso. Además del propio Sprint, que es un contenedor del resto de eventos, cada uno de los eventos de Scrum constituye una oportunidad formal para la inspección y adaptación de algún aspecto. Estos eventos están diseñados específicamente para habilitar las vitales transparencia e inspección. La falta de alguno de estos eventos da como resultado una reducción de la transparencia y constituye una oportunidad perdida para inspeccionar y adaptarse. (Schwaber & Sutherland, 2013, p. 9)

Los eventos están descritos a continuación:

El Sprint

El corazón de Scrum es el Sprint, es un bloque de tiempo (time-box) de un mes o menos durante el cual se crea un incremento de producto "Terminado", utilizable y potencialmente desplegable. Es más conveniente si la duración de los Sprints es consistente a lo largo del esfuerzo de desarrollo. Cada nuevo Sprint comienza inmediatamente después de la finalización del Sprint previo. Los Sprints contienen y consisten de la Reunión de Planificación del Sprint (Sprint Planning Meeting), los Scrums Diarios (Daily Scrums), el trabajo de desarrollo, la Revisión del Sprint (Sprint Review), y la Retrospectiva del Sprint (Sprint Retrospective). (Schwaber & Sutherland, 2013, p. 9)

Según Schwaber & Sutherland (2013), durante el Sprint:

- No se realizan cambios que puedan afectar al Objetivo del Sprint (Sprint Goal);
- Los objetivos de calidad no disminuyen; y,

- El alcance puede ser clarificado y renegociado entre el Dueño de Producto y el Equipo de Desarrollo a medida que se va aprendiendo más.

Cada Sprint puede considerarse un proyecto con un horizonte no mayor de un mes. Al igual que los proyectos, los Sprints se usan para lograr algo. Cada Sprint tiene una definición de qué se va a construir, un diseño y un plan flexible que guiará la construcción y el trabajo y el producto resultante. (Schwaber & Sutherland, 2013, p. 9)

Cancelación de un Sprint

Un Sprint puede ser cancelado antes de que el bloque de tiempo llegue a su fin. Solo el Dueño de Producto tiene la autoridad para cancelar el Sprint, aunque puede hacerlo bajo la influencia de los interesados, del Equipo de Desarrollo o del Scrum Master. Un Sprint se cancelaría si el Objetivo del Sprint llega a quedar obsoleto. Esto podría ocurrir si la compañía cambia la dirección o si las condiciones del mercado o de la tecnología cambian. En general, un Sprint debería cancelarse si no tuviese sentido seguir con él dadas las circunstancias. Pero debido a la corta duración de los Sprints, rara vez la cancelación tiene sentido. (Schwaber & Sutherland, 2013, p. 9)

Cuando se cancela un Sprint, se revisan todos los Elementos de la Lista de Producto que se hayan completado y "Terminado". Si una parte del trabajo es potencialmente entregable, el Dueño de Producto normalmente lo acepta. Todos los Elementos de la Lista de Producto no completados se vuelven a estimar y se vuelven a introducir en la Lista de Producto. El trabajo finalizado en ellos pierde valor con rapidez y frecuentemente debe volverse a estimar. Las cancelaciones de Sprint consumen recursos, ya que todos deben reagruparse en otra Reunión de Planificación de Sprint para empezar otro Sprint. Las cancelaciones de Sprint son a menudo traumáticas para el Equipo Scrum y son muy poco comunes. (Schwaber & Sutherland, 2013, p. 10)

Reunión de Planificación de Sprint (Sprint Planning Meeting)

El trabajo a realizar durante el Sprint se planifica en la Reunión de Planificación de Sprint. Este plan se crea mediante el trabajo colaborativo del Equipo Scrum completo. La Reunión de Planificación de Sprint tiene un máximo de duración de ocho horas para un Sprint de un mes. Para Sprints más cortos, el evento es usualmente más corto. El Scrum Master se asegura de que el evento se lleve a cabo y que los asistentes entiendan

su propósito. El Scrum Master enseña al Equipo Scrum a mantenerse dentro del bloque de tiempo. (Schwaber & Sutherland, 2013, p. 10)

Según Schwaber & Sutherland (2013), la Reunión de Planificación de Sprint responde a los siguientes interrogantes:

- ¿Qué puede entregarse en el Incremento resultante del Sprint que comienza?
- ¿Cómo se conseguirá hacer el trabajo necesario para entregar el Incremento?

Para saber qué puede ser terminado en un Sprint, según redactan Schwaber & Sutherland (2013), el Equipo de Desarrollo trabaja para proyectar la funcionalidad que se desarrollará durante el Sprint. El Dueño de Producto discute el objetivo que el Sprint debería lograr y los Elementos de la Lista de Producto que, si se completan en el Sprint, lograrían el Objetivo del Sprint. El Equipo Scrum completo colabora en el entendimiento del trabajo del Sprint. (p. 10)

La entrada a esta reunión está constituida por la Lista de Producto, el último Incremento de producto, la capacidad proyectada del Equipo de Desarrollo para el Sprint, y el rendimiento pasado del Equipo de Desarrollo. El número de elementos de la Lista de Producto seleccionados para el Sprint depende únicamente del Equipo de Desarrollo. Solo el Equipo de Desarrollo puede evaluar qué es capaz de lograr durante el Sprint que comienza. Después de que el Equipo de Desarrollo proyecta qué elementos de la Lista de Producto entregará en el Sprint, el Equipo Scrum elabora un Objetivo del Sprint (Sprint Goal). El Objetivo del Sprint debería lograrse durante el Sprint a través de la implementación de la Lista de Producto, y provee una guía al equipo de desarrollo de por qué se está construyendo el incremento. (Schwaber & Sutherland, 2013, p. 11)

Para saber como se conseguirá completar el trabajo seleccionado, según Schwaber & Sutherland (2013), Una vez que se ha establecido el objetivo y seleccionado los elementos de la Lista de Producto para el Sprint, el Equipo de Desarrollo decide cómo construirá esta funcionalidad para formar un Incremento de producto "Terminado". Los elementos de la Lista de Producto seleccionados para este Sprint, más el plan para terminarlos, recibe el nombre de Lista de Pendientes del Sprint (Sprint Backlog).

El Equipo de Desarrollo por lo general comienza diseñando el sistema y el trabajo necesario para convertir la Lista de Producto en un Incremento de producto funcional. El trabajo podría ser de tamaño o esfuerzo estimado variables. Sin embargo, durante la Reunión de Planificación del Sprint, se planifica suficiente trabajo como para que el Equipo de Desarrollo pueda hacer una proyección de lo que cree que puede completar en el Sprint que comienza. Para el final de esta reunión, el trabajo planificado por el Equipo de Desarrollo para los primeros días del Sprint es descompuesto en unidades de un día o menos. El Equipo de desarrollo se autoorganiza para asumir el trabajo de la Lista de Pendientes de Sprint, tanto durante la reunión de Planificación de Sprint como a lo largo del Sprint. (Schwaber & Sutherland, 2013, p. 11)

Al finalizar la Reunión de Planificación de Sprint, el Equipo de Desarrollo debería ser capaz de explicar al Dueño de Producto y al Scrum Master cómo pretende trabajar como un equipo autoorganizado para lograr el Objetivo del Sprint y crear el Incremento esperado. (Schwaber & Sutherland, 2013, p. 11)

Objetivo del Sprint (Sprint Goal)

El Objetivo del Sprint es una meta establecida para el Sprint que puede ser alcanzada mediante la implementación de la Lista de Producto. Proporciona una guía al Equipo de Desarrollo acerca de por qué está construyendo el incremento. Es creado durante la reunión de Planificación del Sprint. El objetivo del Sprint ofrece al equipo de desarrollo cierta flexibilidad con respecto a la funcionalidad implementada en el Sprint. Los elementos de la Lista del Producto seleccionados ofrecen una función coherente, que puede ser el objetivo del Sprint. El objetivo del Sprint puede representar otro nexo de unión que haga que el Equipo de Desarrollo trabaje en conjunto y no en iniciativas separadas. (Schwaber & Sutherland, 2013, p. 12)

Scrum Diario (Daily Scrum)

El Scrum Diario es una reunión con un bloque de tiempo de 15 minutos para que el Equipo de Desarrollo sincronice sus actividades y cree un plan para las siguientes 24 horas. Esto se lleva a cabo inspeccionando el trabajo avanzado desde el último Scrum Diario y haciendo una proyección acerca del trabajo que podría completarse antes del siguiente. (Schwaber & Sutherland, 2013, p. 12)

Según Schwaber & Sutherland (2013), durante la reunión cada miembro del Equipo de Desarrollo explica:

- ¿Qué hice ayer que ayudó al Equipo de Desarrollo a lograr el Objetivo del Sprint?
- ¿Qué haré hoy para ayudar al Equipo de Desarrollo a lograr el Objetivo del Sprint?
- ¿Veo algún impedimento que evite que el Equipo de Desarrollo o yo logremos el Objetivo del Sprint?

El Equipo de Desarrollo usa el Scrum Diario para evaluar el progreso hacia el Objetivo del Sprint y para evaluar qué tendencia sigue este progreso hacia la finalización del trabajo contenido en la Lista del Sprint. El Scrum Diario optimiza las posibilidades de que el Equipo de Desarrollo cumpla el Objetivo del Sprint. Cada día, el Equipo de Desarrollo debería entender cómo intenta trabajar en conjunto como un equipo autoorganizado para lograr el Objetivo del Sprint y crear el Incremento esperado hacia el final del Sprint. El Equipo de Desarrollo o los miembros del equipo a menudo se vuelven a reunir inmediatamente después del Scrum Diario, para tener discusiones detalladas, o para adaptar, o replanificar el resto del trabajo del Sprint. (Schwaber & Sutherland, 2013, p. 12)

Los Scrum Diarios mejoran la comunicación, eliminan la necesidad de mantener otras reuniones, identifican y eliminan impedimentos relativos al desarrollo, resaltan y promueven la toma de decisiones rápida, y mejoran el nivel de conocimiento del Equipo de Desarrollo. El Scrum Diario constituye una reunión clave de inspección y adaptación. (Schwaber & Sutherland, 2013, p. 13)

Revisión de Sprint (Sprint Review)

Se trata de una reunión restringida a un bloque de tiempo de cuatro horas para Sprints de un mes. Para Sprints más cortos, se reserva un tiempo proporcionalmente menor. El Scrum Master se asegura de que el evento se lleve a cabo y que los asistentes entiendan su propósito. El Scrum Master enseña a todos a mantener el evento dentro del bloque de tiempo fijado. (Schwaber & Sutherland, 2013, p. 13)

Según Schwaber & Sutherland (2013), la Revisión de Sprint incluye los siguientes elementos:

- Los asistentes son el Equipo Scrum y los interesados clave invitados por el Dueño de Producto;
- El Dueño de Producto explica qué elementos de la Lista de Producto se han "Terminado" y cuales no se han "Terminado";
- El Equipo de Desarrollo habla acerca de qué fue bien durante el Sprint, qué problemas aparecieron y cómo fueron resueltos esos problemas;
- El Equipo de Desarrollo demuestra el trabajo que ha "Terminado" y responde preguntas acerca del Incremento;
- El Dueño de Producto habla acerca de la Lista de Producto en el estado actual. Proyecta fechas de finalización probables en el tiempo basándose en el progreso obtenido hasta la fecha (si es necesario);
- El grupo completo colabora acerca de qué hacer a continuación, de modo que la Revisión del Sprint proporcione información de entrada valiosa para Reuniones de Planificación de Sprints subsiguientes.
- Revisión de cómo el mercado o el uso potencial del producto podría haber cambiado lo que es de más valor para hacer a continuación; y,
- Revisión de la línea de tiempo, presupuesto, capacidades potenciales y mercado para la próxima entrega prevista del producto.

El resultado de la Revisión de Sprint es una Lista de Producto revisada, que define los elementos de la Lista de Producto posibles para el siguiente Sprint. Es posible además que la Lista de Producto reciba un ajuste general para enfocarse en nuevas oportunidades. (Schwaber & Sutherland, 2013, p. 14)

Retrospectiva de Sprint (Sprint Retrospective)

La Retrospectiva de Sprint es una oportunidad para el Equipo Scrum de inspeccionarse a sí mismo y crear un plan de mejoras que sean abordadas durante el siguiente Sprint. La Retrospectiva de Sprint tiene lugar después de la Revisión de Sprint y antes de la siguiente Reunión de Planificación de Sprint. Se trata de una reunión restringida a un bloque de tiempo de tres horas para Sprints de un mes. Para Sprints más cortos se reserva un tiempo proporcionalmente menor. El Scrum Master se asegura

de que el evento se lleve a cabo y que los asistentes entiendan su propósito. El Scrum Master enseña a todos a mantener el evento dentro del bloque de tiempo fijado. El Scrum Master participa en la reunión como un miembro del equipo ya que la responsabilidad del proceso Scrum recae sobre él. (Schwaber & Sutherland, 2013, p. 14)

Según Schwaber & Sutherland (2013), el propósito de la Retrospectiva de Sprint es:

- Inspeccionar cómo fue el último Sprint en cuanto a personas, relaciones, procesos y herramientas;
- Identificar y ordenar los elementos más importantes que salieron bien y las posibles mejoras; y,
- Crear un plan para implementar las mejoras a la forma en la que el Equipo Scrum desempeña su trabajo.

Artefactos de Scrum

Los artefactos de Scrum representan trabajo o valor en diversas formas que son útiles para proporcionar transparencia y oportunidades para la inspección y adaptación. Los artefactos definidos por Scrum están diseñados específicamente para maximizar la transparencia de la información clave, que es necesaria para asegurar que todos tengan el mismo entendimiento del artefacto. (Schwaber & Sutherland, 2013, p. 15)

Lista de Producto (Product Backlog)

La Lista de Producto es una lista ordenada de todo lo que podría ser necesario en el producto, y es la única fuente de requisitos para cualquier cambio a realizarse en el producto. El Dueño de Producto (Product Owner) es el responsable de la Lista de Producto, incluyendo su contenido, disponibilidad y ordenación. Una Lista de Producto nunca está completa. El desarrollo más temprano de la misma solo refleja los requisitos conocidos y mejor entendidos al principio. La Lista de Producto evoluciona a medida de que el producto y el entorno en el que se usará también lo hacen. La Lista de Producto es dinámica; cambia constantemente para identificar lo que el producto necesita para ser adecuado, competitivo y útil. Mientras el producto exista, su Lista de Producto también existe. (Schwaber & Sutherland, 2013, p. 15)

La Lista de Producto enumera todas las características, funcionalidades, requisitos, mejoras y correcciones que constituyen cambios a ser hechos sobre el producto para entregas futuras. Los elementos de la Lista de Producto tienen como atributos la descripción, la ordenación, la estimación y el valor. (Schwaber & Sutherland, 2013, p. 15)

Existe un refinamiento (refinement) de la Lista de Producto es el acto de añadir detalle, estimaciones y orden a los elementos de la Lista de Producto. Se trata de un proceso continuo, en el cual el Dueño de Producto y el Equipo de Desarrollo colaboran acerca de los detalles de los elementos de la Lista de Producto. Durante el refinamiento de la Lista de Producto, se examinan y revisan sus elementos. El Equipo Scrum decide cómo y cuándo se hace el refinamiento. Este usualmente consume no más del 10% de la capacidad del Equipo de Desarrollo. Sin embargo, los elementos de la Lista de Producto pueden actualizarse en cualquier momento por el Dueño de Producto o a criterio suyo. (Schwaber & Sutherland, 2013, p. 15)

Seguimiento del Progreso Hacia un Objetivo

En cualquier momento, es posible sumar el trabajo total restante para alcanzar el objetivo. El Dueño de Producto hace seguimiento de este trabajo restante total al menos en cada Revisión de Sprint. El Dueño de Producto compara esta cantidad con el trabajo restante en Revisiones de Sprint previas, para evaluar el progreso hacia la finalización del trabajo proyectado en el tiempo deseado para el objetivo. Esta información se muestra de forma transparente a todos los interesados. (Schwaber & Sutherland, 2013, p. 16)

Lista de Pendientes del Sprint (Sprint Backlog)

La Lista de Pendientes del Sprint es el conjunto de elementos de la Lista de Producto seleccionados para el Sprint, más un plan para entregar el Incremento de producto y conseguir el Objetivo del Sprint. La Lista de Pendientes del Sprint es una predicción hecha por el Equipo de Desarrollo acerca de qué funcionalidad formará parte del próximo Incremento y del trabajo necesario para entregar esa funcionalidad en un Incremento "Terminado". La Lista de Pendientes del Sprint hace visible todo el trabajo

que el Equipo de Desarrollo identifica como necesario para alcanzar el Objetivo del Sprint. (Schwaber & Sutherland, 2013, p. 16)

Seguimiento del Progreso del Sprint

En cualquier momento durante un Sprint, es posible sumar el trabajo restante total en los elementos de la Lista de Pendientes del Sprint. El Equipo de Desarrollo hace seguimiento de este trabajo restante total al menos en cada Scrum Diario para proyectar la posibilidad de conseguir el Objetivo del Sprint. Haciendo seguimiento del trabajo restante a lo largo del Sprint, el Equipo de Desarrollo puede gestionar su progreso. (Schwaber & Sutherland, 2013, p. 17)

Incremento

El Incremento es la suma de todos los elementos de la Lista de Producto completados durante un Sprint y el valor de los incrementos de todos los Sprints anteriores. Al final de un Sprint, el nuevo Incremento debe estar "Terminado", lo cual significa que está en condiciones de ser utilizado y que cumple la Definición de "Terminado" del Equipo Scrum. El incremento debe estar en condiciones de utilizarse sin importar si el Dueño de Producto decide liberarlo o no. (Schwaber & Sutherland, 2013, p. 17)

Transparencia de los Artefactos

Scrum se basa en la transparencia. Las decisiones para optimizar el valor y controlar el riesgo se toman basadas en el estado percibido de los artefactos. En la medida en que la transparencia sea completa, estas decisiones tienen unas bases sólidas. En la medida en que los artefactos no son completamente transparentes, estas decisiones pueden ser erróneas, el valor puede disminuir y el riesgo puede aumentar. El Scrum Master debe trabajar con el Dueño de Producto, el Equipo de Desarrollo y otras partes involucradas para entender si los artefactos son completamente transparentes. Hay prácticas para hacer frente a la falta de transparencia; el Scrum Master debe ayudar a todos a aplicar las prácticas más apropiadas si no hay una transparencia completa. Un Scrum Master puede detectar la falta de transparencia inspeccionando artefactos, reconociendo patrones, escuchando atentamente lo que se dice y detectando diferencias entre los resultados esperados y los reales; la labor del Scrum Master es trabajar con el Equipo

Scrum y la organización para mejorar la transparencia de los artefactos. Este trabajo usualmente incluye aprendizaje, convicción y cambio. (Schwaber & Sutherland, 2013, p. 17)

Para terminar con los conceptos de la metodología Scrum, se procede a definir la palabra "Terminado" que mucha veces fue utilizada a lo largo de la descripción de esta metodología.

Definición de "Terminado" (Definition of "Done")

Es un término que todos los participantes o asociados al proyecto deben entender de la misma manera cuando se refieran a que el avance o trabajo se encuentre en estado completado.

Cuando un elemento de la Lista de Producto o un Incremento se describe como "Terminado", todo el mundo debe entender lo que significa "Terminado". Aunque esto varía significativamente para cada Equipo Scrum, los miembros del Equipo deben tener un entendimiento compartido de lo que significa que el trabajo esté completado, para asegurar la transparencia. Esta es la definición de "Terminado" para el Equipo Scrum y se utiliza para evaluar cuándo se ha completado el trabajo sobre el Incremento de producto. Si la definición de "Terminado" para un incremento es parte de las convenciones, estándares o guías de la organización de desarrollo, al menos todos los Equipos Scrum deben seguirla. Si "Terminado" para un incremento no es una convención de la organización de desarrollo, el Equipo de Desarrollo del Equipo Scrum debe definir una definición de "Terminado" apropiada para el producto. Si hay múltiples Equipos Scrum trabajando en la entrega del sistema o producto, los equipos de desarrolladores en todos los Equipos Scrum deben definir en conjunto la definición de "Terminado". (Schwaber & Sutherland, 2013, p. 18)

➢ **Metodologías Crystal:**

Se trata de un conjunto de metodologías para el desarrollo de software caracterizadas por estar centradas en las personas que componen el equipo y la reducción al máximo del número de artefactos producidos. El desarrollo de software se considera un juego cooperativo de invención y comunicación, limitado por los recursos a utilizar. El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo definidas. (Orjuela Duarte & Rojas, según citan de Fowler, 2008, p. 162).

Orjuela Duarte & Rojas (2008) describen una serie de características, que son presentadas a continuación:

- Cuando el número de personas aumenta, también aumenta la necesidad de coordinar.
- Cuando el potencial de daño se incrementa, la tolerancia a variaciones se ve afectada.
- La sensibilidad del tiempo en que se debe estar en el mercado varía: a veces este tiempo debe acortarse al máximo y se toleran defectos, otras se enfatiza la auditoría, confiabilidad, protección legal, entre otros.
- Las personas se comunican cara a cara, con la pregunta y la respuesta en el mismo espacio de tiempo.
- El factor más significativo es la comunicación.

Crystal cuenta con cuatro variantes de metodologías: Crystal Clear para equipos conformados por 8 o menos integrantes; Amarillo, para equipos conformados entre 8 a 20 integrantes; Naranja, para equipos conformados entre 20 a 50 integrantes; y Rojo, para equipos conformados entre 50 a 100 integrantes. (Orjuela Duarte & Rojas, 2008, p. 165)

Por motivos de poca documentación y del tamaño del equipo, que se adapta a las metodologías que se contextualizan con este proyecto, solo se va a describir la versión Crystal Clear.

Crystal Clear

Metodología que se adapta a proyectos conformados por equipos pequeños (ver *Figura 2.7*).

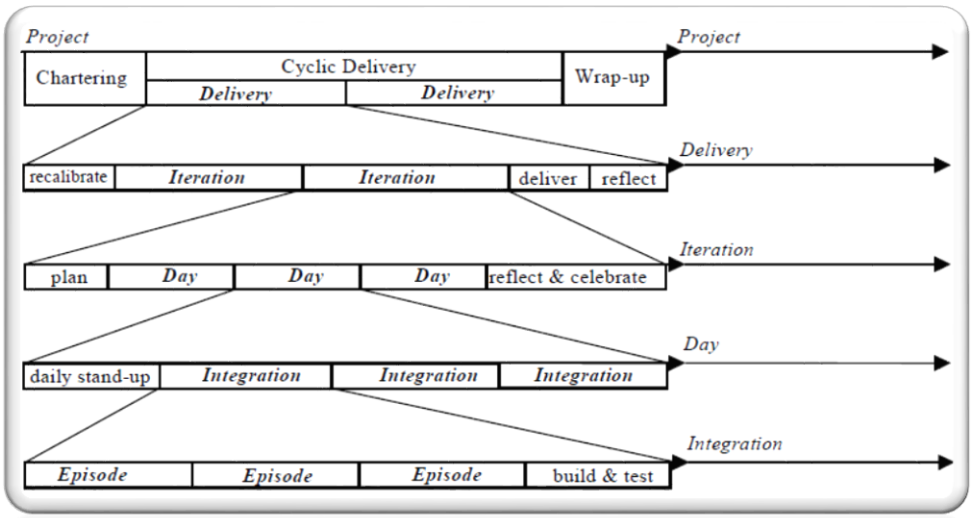


Figura 2.7. Proceso Crystal Clear.
Ramsin, (s.f). Agile Methodologies: Crystal.

Según Orjuela Duarte & Rojas (2008), Crystal Clear consiste de una serie de valores y técnicas que describen de la siguiente manera:

Valores:

- Entrega frecuente. Consiste en hacer entrega de software a los clientes con frecuencia; la frecuencia depende el proyecto, pero puede ser diaria, semanal o mensual.
- Comunicación osmótica. Todos juntos en el mismo cuarto; una variante especial, es disponer en la sala de un experto diseñador senior y discutir del tema que se trate.

- Mejora reflexiva. Tomarse un pequeño tiempo (unas pocas horas, cada o una vez al mes) para pensar bien que se está haciendo, cotejar notas, reflexionar, discutir, etc.
- Seguridad personal. Dialogar con los compañeros cuando algo molesta dentro del grupo.
- Foco. Saber lo que se está haciendo y tener la tranquilidad y el tiempo para hacerlo.
- Fácil acceso a usuarios expertos. Tener alguna comunicación con expertos desarrolladores.

Técnicas

- Exploración de 360°. Verificar o tomar una muestra del valor de negocios del proyecto, los requerimientos, el modelo de dominio, la tecnología, el plan del proyecto y el proceso.
- Victoria temprana. Es mejor buscar pequeños triunfos iniciales que aspirar a una gran victoria tardía.
- Esqueleto ambulante. Es una transacción que debe ser simple pero completa.
- Reaquitectura incremental. No es conveniente interrumpir el desarrollo para corregir la arquitectura, la arquitectura debe evolucionar en etapas, manteniendo el sistema en ejecución mientras ella se modifica.
- Radiadores de información. Es una lámina que contiene anuncios o novedades producidas en el entorno del equipo de trabajo, pegada en algún lugar visible o que el equipo pueda observar mientras trabaja o camina. Tiene que ser comprensible para el trabajador casual, entendida de un vistazo y renovada periódicamente para que valga la pena visitarla.

En el contexto de las técnicas se favorecen:

- Entrevistas de proyectos. Se suele entrevistar a más de un responsable para tener visiones más ricas.
- Talleres de reflexión. El equipo debe detenerse treinta minutos o una hora para reflexionar sobre sus convenciones de trabajo, discutir inconvenientes y mejoras y planear para el periodo siguiente.

- Planteamiento Blitz. Una técnica puede ser el juego de planteamiento de XP, en este juego se ponen tarjetas indexadas en una mesa, con una historia de usuario o función visible en cada una. El grupo finge que no hay dependencia entre tarjetas, y las alinea en secuencia de desarrollo preferida. Los desarrolladores escriben en cada tarjeta el tiempo estimado para desarrollar cada función. El patrocinador del usuario escribe la secuencia de prioridades, teniendo en cuenta los tiempos referidos y el valor de negocio de cada función. Las tarjetas se agrupan en periodos de tres semanas llamados iteraciones que se agrupan en entregas, usualmente no más largas de tres meses.
- Estimación Delphi con estimaciones de pericia. En el modelo Delphi se reúnen los expertos responsables y proceden como en un remate para proponer el tamaño del sistema, su tiempo de ejecución, la fecha de las entregas según dependencias técnicas y de negocios y para equilibrar en paquetes de igual tamaño.
- Encuentros diarios de pie. La palabra clave es "brevedad", 5 a 10 minutos como máximo. No se trata de discutir problemas, sino de identificarlos.
- Miniatura de procesos. Una forma de presentar Crystal Clear puede consumir entre 90 minutos y un día. La idea es que las personas puedan degustar la metodología.
- Gráficos de quemado. Se trata de una técnica de graficación para descubrir demoras y problemas tempranamente en el proceso, evitando que se descubra demasiado tarde y sin saber todavía cuánto falta. Para ello se hace una estimación del tiempo faltante para desarrollar lo que resta al ritmo actual, que sirve para tener dominio de proyectos en los cuales las prioridades cambian bruscamente y con frecuencia. Esta técnica se asocia con algunos recursos ingeniosos, como la Lista Témpana, llamada así porque se refiere al agregado de ítems con alta prioridad en el tope de las listas de trabajo pendiente, esperando que los demás elementos se hundan bajo la línea de flotación; los elementos que están sobre la línea se entregarán en la iteración siguiente, los que están por debajo en las iteraciones restantes. Los gráficos de quemado ilustran la velocidad del proceso, analizando la diferencia entre las líneas proyectadas y efectivas de cada entrega.

- Programación lado a lado. Crystal Clear establece proximidad, pero cada desarrollador se enfoca a su trabajo asignado, prestando un poco de atención a lo que hace el compañero que tiene a su lado, cada uno con su propia terminal de trabajo.

Crystal Clear contiene unos roles nominados: Patrocinador, Usuario Experto, Diseñador Principal, Diseñador Programador, Experto en Negocios, Coordinador, Verificador y Escritor. En Crystal Naranja se agregan aún más roles: Diseñador UI (Interfaz de Usuario), Diseñador de Base de Datos, Experto en Uso, Facilitador Técnico, Analista/Diseñador de Negocios, Arquitecto, Mentor de Diseño y Punto de Reutilización. (Orjuela Duarte & Rojas, 2008, p. 166)

Orjuela Duarte & Rojas (2008), describen los artefactos de los que son responsables los roles de Crystal Clear de la siguiente manera:

Patrocinador

Produce la Declaración de Misión con Prioridades de Compromiso. Consigue los recursos y define la totalidad del proyecto.

Usuario Experto

Junto con el Experto en Negocios produce la Lista de Actores Objetivos y el Archivo de Casos de Uso y Requerimientos. Debe familiarizarse con el uso del sistema, sugerir atajos de teclado, modos de operación, información a visualizar simultáneamente y navegación.

Diseñador Principal

Produce la Descripción Arquitectónica. Se supone que debe ser un profesional de nivel 3 (en metodologías ágiles se definen tres niveles de experiencia: Nivel 1 es capaz de seguir los procedimientos; Nivel 2 es capaz de apartarse de los procedimientos específicos y encontrar otros distintos; Nivel 3 es capaz de manejar con fluidez, mezclar en inventar procedimientos). El Diseñador Principal tiene roles de coordinador, arquitecto, mentor y desarrollador más experto.

Diseñador Programador (Desarrollador)

Produce junto con el Diseñador Principal, los Borradores de Pantallas, el Modelo Común de Dominio, las Notas y Diagramas de Diseño, el Código Fuente, el Código de Migración, las Pruebas y el Sistema Empaquetado. Un programa en Crystal Clear es diseño y programa; sus desarrolladores son diseñadores-desarrolladores; un diseñador en esta metodología de desarrollo debe saber programar.

Experto en Negocios

Junto con el Usuario Experto produce la lista de Actores Objetivos y el Archivo de Casos de Uso y Requerimientos. Debe conocer las reglas y políticas del negocio.

Coordinador

Con la ayuda del equipo, produce el Mapa de Proyecto, el Plan de Entrega, el Estado del Proyecto, la Lista de Riesgos, el Plan y Estado de Iteración y la Agenda de Visualización.

Verificador

Produce el Reporte de Bugs. Puede ser un desarrollador en tiempo parcial, o un equipo de varias personas.

Escritor

Produce el Manual de Usuario. El equipo como grupo es responsable de producir la estructura y convenciones del equipo y los resultados del taller de reflexión.

➢ Metodología Kanban

Kanban es una metodología enfocada en la transformación gradual y continua de las organizaciones en organismos eficientes, adaptativos y centrados en el valor para su cliente. Sus prácticas de visualización y gestión del flujo de trabajo, y de mejora continua ayudan a acelerar el desarrollo de los proyectos y las operaciones TI (Tecnologías de Información), reducir el tiempo de entrega, optimizar la utilización de recursos, eliminar los desperdicios en los procesos, aumentar la calidad de los productos y los servicios, establecer procesos sostenibles y rentables, y fortalecer la colaboración entre los implicados (ver *Figura 2.8*). (Bozheva de Berriprocess, 2013, p. 494)

Según Bozheva de Berriprocess (2013), que cita a David Anderson, Kanban es un conjunto de unos principios y unas series de prácticas entrelazadas a través de los valores de una organización ágil, enfocada en la creación de valor para sus clientes, evolucionando continuamente.

Figura 2.8. Proceso Kanban.
Gamboa Manzaba, (2014). Aumento de la productividad en la gestión de proyectos, utilizando una metodología ágil aplicada en una fábrica de software en la ciudad de Guayaquil.

A continuación se describen cuales son esos principios, prácticas y valores:

Principios

Los principios que destaca Bozheva de Berriprocess (2013) son:

- Animar el Liderazgo.
- Inicialmente Respetar Procesos y Responsabilidades.
- Comprometerse a Buscar Cambios Incrementales y Evolutivos.
- Empezar con las Prácticas Actuales.

Valores

Los valores que destaca Bozheva de Berriprocess (2013) son:

- Liderazgo.
- Transparencia.
- Flujo.
- Enfoque en el cliente.
- Acuerdo.
- Colaboración.
- Entendimiento.
- Respeto.
- Equilibrio.

Prácticas

Las prácticas Visualizar el Flujo de Trabajo, Limitar el Proceso en Curso, Gestionar el Flujo, Establecer Políticas de Calidad, las Prácticas Organizativas y Mejorar Colaborando destacadas por Bozheva de Berriprocess (2013) son descritas a continuación:

Visualizar el Flujo de Trabajo

El propósito principal de Kanban es optimizar los procesos actuales. Visualizar el flujo de trabajo actual (no el oficial que podría existir pero no usarse) significa que

todos los involucrados pueden ver lo que está en curso y cómo se hace, y es esencial para el entendimiento de los procesos. El flujo de trabajo se visualiza a través de un tablero… Las columnas representan las fases por las que pasa el producto en su proceso de evolución. Las filas pueden representar o bien distintos tipos de trabajo (desarrollo nuevo, cambios a los requisitos, corrección de incidencias, tareas urgentes, etc.) o bien diferentes proyectos que se ejecutan en paralelo. Cada tarjeta Kanban muestra la tarea que se está ejecutando, la persona asignada, fechas importantes de la tarea, prioridad, si está retrasada o bloqueada, etc. (Bozheva de Berriprocess, 2013, p. 492)

Un tablero Kanban pone a la vista la información necesaria para controlar la evolución del producto e involucrar de forma natural a todos los integrantes en la resolución y la gestión de los problemas. Estas características suyas lo convierten en el punto principal de información sobre el proyecto y el facilitador de la colaboración y la coordinación entre las personas. El hecho de tener continuamente visibilidad sobre el estado del trabajo concentra las acciones de gestión directamente en la resolución de los asuntos bloqueantes que se han demostrado en el tablero y elimina el tiempo y el esfuerzo para identificarlos y localizarlos. Este es uno de los factores que llevan a la aceleración de los desarrollos y los servicios, y a la reducción del coste de coordinación. (Bozheva de Berriprocess, 2013, p. 492)

"La ejecución de esta práctica está acorde con dos **principios**, empezar con las prácticas actuales e inicialmente, respetar procesos, responsabilidades y cargos actuales. Los **valores** organizativos que apoyan su aplicación correcta son entendimiento, acuerdo y respeto". (Bozheva de Berriprocess, 2013, p. 492)

Limitar el Trabajo en Curso

Los Límites de Trabajo en Curso es el número máximo de tarjetas o partes que se pueden trabajar en un determinado transcurso de tiempo.

Los Límites de Trabajo en Curso, en cada una de las fases y los proyectos están mostrados a través de los números que acompañan los nombres de las columnas, generalmente entre paréntesis. El número de tarjetas en la columna no puede superar el límite establecido, es decir no se puede asumir más trabajo por los recursos que realizan las tareas que corresponden a la columna. Para que no pare el flujo completo,

los recursos que están libres, pero no pueden "tirar" más tareas, tienen que ayudar para que avance el trabajo en su fase (columna), en las próximas fases, o realizar otras tareas útiles sin añadir más trabajo en curso. (Bozheva de Berriprocess, 2013, p. 493)

Según Bozheva de Berriprocess (2013), limitar el trabajo en curso tiene varios aspectos positivos:

- Se presta más atención al trabajo que está en curso, por lo que terminarlo correctamente y con mayor calidad tiene una prioridad más alta.
- Reduce las colas de trabajo cumplido que está esperando a ser cogido (pulled) por la próxima fase en el flujo, p.ej. código programado que está esperando ser probado.
- Facilita la gestión de los cuellos de botella en el flujo de trabajo.
- Reduce la pérdida de productividad debido a los cambios de tareas sobre las que se trabaja. Estudios de este tema demuestran que cada vez que cambiamos de tarea, aumentamos el tiempo de su ejecución con un 20%.

Gestionar el Flujo

Kanban está enfocado en establecer un flujo de trabajo eficiente continuo, suave y previsible, adaptar la utilización de los recursos a la demanda y aumentar la calidad de los productos/servicios. Por lo tanto, las métricas que utiliza son un poco diferentes de las que estamos acostumbrados a manejar. (Bozheva de Berriprocess, 2013, p. 493)

Según Bozheva de Berriprocess (2013), los aspectos fundamentales que se gestionan son los siguientes:

- El trabajo en curso, a través de un gráfico de área que representa la cantidad de trabajo en un estado determinado, mostrando las llegadas de las peticiones de trabajo, tiempo en cola, cantidad en la cola, y la salida.
- El tiempo de entrega, típicamente a través de un histograma, gráfica de control o de cajas, para conocerlo en base a datos reales y poder mejorarlo.
- El rendimiento, habitualmente a través de una gráfica de barras, como un indicador de cómo funciona el sistema y el nivel de mejora conseguido.

- Asuntos bloqueantes y la rapidez de su resolución, de nuevo a través de un diagrama de flujo acumulado; no obstante, esta vez controlando el número de asuntos abiertos, sobre los que se está trabajando y los cerrados. Entender las causas principales y/o raíz para los asuntos bloqueantes, así como su impacto sobre el flujo completo de trabajo, es fundamental para eliminar los impedimentos, acelerar los desarrollos y los servicios, y aumentar la agilidad del negocio.
- Calidad del desarrollo o el servicio. El número de defectos que se han escapado y han llegado al cliente representan el nivel de calidad y también afectan al tiempo de entrega y al rendimiento del sistema, dado que su entrada inesperada en éste aumenta y relentiza la finalización del trabajo que ya está en curso. En general, el objetivo es garantizar que la tasa de defectos es decreciente.

Establecer Políticas de Calidad

El propósito principal de las Políticas de Calidad en Kanban es asegurar que el producto que se está desarrollando en el flujo de trabajo tiene buen nivel de calidad en cada fase del flujo. En términos prácticos las políticas de calidad se pueden considerar como criterios que aseguran la coherencia de las actividades y que se deben cumplir para dar el producto por terminado en una fase y poder pasarlo a la próxima. La visualización de las políticas es un factor importante que facilita su aplicación. (Bozheva de Berriprocess, 2013, p. 493)

Prácticas Organizativas

La visión de una organización se consigue en pasos pequeños, eliminando los obstáculos entre el estado actual y el estado objetivo de la organización. Las últimas dos **prácticas**, Establecer Mecanismos de Retroalimentación y Mejorar Colaborando, tratan de ampliar la aplicación del método a nivel de organización y establecer la rutina de resolución de problemas y optimización continua del flujo como una forma de mejora tanto de los productos y los servicios, como de los resultados económicos de una empresa. (Bozheva de Berriprocess, 2013, p. 494)

Mejorar Colaborando

La colaborción es una de las formas más efectivas de generar buena relación entre los integrantes de los equipos de trabajo, y consecuentemente los avances o mejoras de cada uno de los integrantes son evidentes dando como resultado productos o servicios de calidad.

➢ **Metodología Mobile-D**

Es una metodología ágil de desarrollo de software para desarrollar aplicaciones dirigidas a dispositivos móviles o en su defecto, plataformas móviles.

"Mobile-D consta de cinco fases: exploración, iniciación, producción, estabilización y prueba del sistema (ver *Figura 2.9*). Cada una de estas fases tiene un número de etapas, tareas y prácticas asociadas". (Amaya Balaguera, 2013, p. 118)

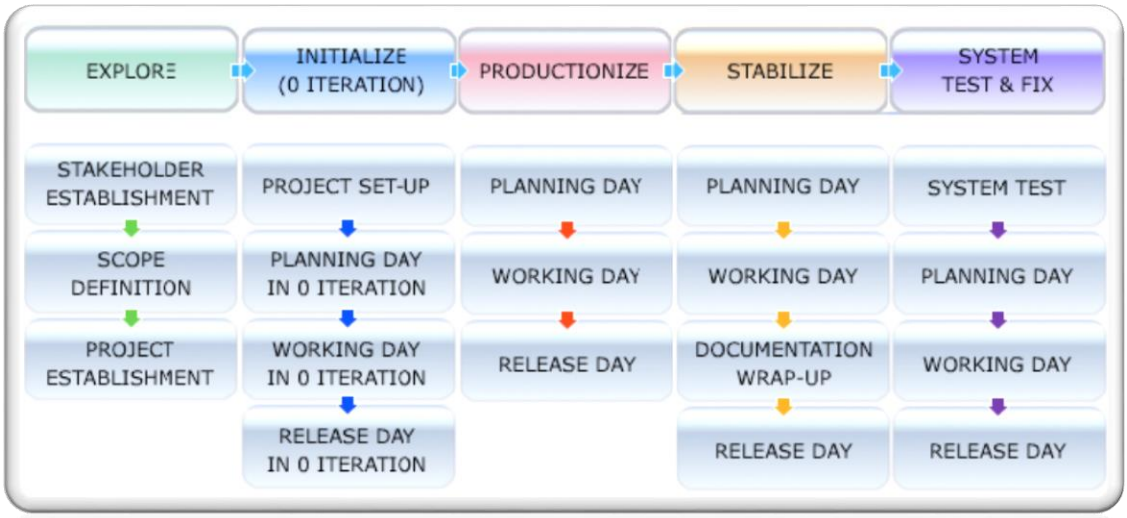


Figura 2.9. Proceso Mobile-D.
Blanco & otros, (2009). Metodología de desarrollo ágil para sistemas móviles. Introducción al desarrollo con Android y el iPhone

A continuación se describen cada una de las fases que componen la metodología de desarrollo Mobile-D:

Exploración

La fase de Exploración, siendo ligeramente diferente del resto del proceso de producción, se dedica al establecimiento de un plan de proyecto y los conceptos básicos; por lo tanto, se puede separar del ciclo principal de desarrollo (aunque no debería obviarse). Los autores de la metodología ponen además especial atención a la

participación de los clientes en esta fase. (Blanco, Camarero, Fumero, Werterski, & Rodríguez, 2009, p. 14)

Iniciación

Durante la fase de Iniciación, los desarrolladores preparan e identifican todos los recursos necesarios. Se preparan los planes para las siguientes fases y se establece el entorno técnico (incluyendo el entrenamiento del equipo de desarrollo). Los autores de Mobile-D afirman que su contribución al desarrollo ágil se centra fundamentalmente en esta fase, en la investigación de la línea arquitectónica. Esta acción se lleva a cabo durante el día de planificación. Los desarrolladores analizan el conocimiento y los patrones arquitectónicos utilizados en la empresa (extraidos de proyectos anteriores) y los relacionan con el proyecto actual. Se agregan las observaciones, se identifican similitudes y se extraen soluciones viables para su aplicación en el proyecto. Finalmente, la metodología también contempla algunas funcionalidades nucleares que se desarrollan en esta fase, durante el día de trabajo. (Blanco, Camarero, Fumero, Werterski, & Rodríguez, 2009, p. 14)

Producción

En la fase de Producción se repite la programación de tres días (planificación-trabajo-liberación) se repite iterativamente hasta implementar todas las funcionalidades. Primero se planifica la iteración de trabajo en términos de requisitos y tareas a realizar. Se preparan las pruebas de la iteración de antemano (de ahí el nombre de esta técnica de Test Driven Development, TDD). Las tareas se llevarána a cabo durante el día de trabajo, desarrollando e integrando el código con los repositorios existentes. Durante el último día se lleva a cabo la integración del sistema (en caso de que estuvieran trabajando varios equipos de forma independiente) seguida de las pruebas de aceptación. (Blanco, Camarero, Fumero, Werterski, & Rodríguez, 2009, p. 14)

Estabilización

En la fase de Estabilización, se llevan a cabo las últimas acciones de integración para asegurar que el sistema completo funciona correctamente. Esta será la fase más importante en los proyectos multi-equipo con diferentes subsistemas desarrollados por

equipos distintos. En esta fase, los desarrolladores realizarán tareas similares a las que debían desarrollar en la fase de Productización, aunque en este caso todo el esfuerzo se dirige a la integración del sistema. Adicionalmente se puede considerar en esta fase la producción de documentación. (Blanco, Camarero, Fumero, Werterski, & Rodríguez, 2009, p. 15)

Prueba del Sistema

"La última fase (Prueba del Sistema) tiene como meta la disponibilidad de una versión estable y plenamente funcional del sistema. El producto terminado e integrado se prueba con los requisitos de cliente y se eliminan todos los defectos encontrados". (Blanco, Camarero, Fumero, Werterski, & Rodríguez, 2009, p. 15)

2.1.3 CARACTERÍSTICAS DESTACADAS DE LAS METODOLOGÍAS ÁGILES ESTUDIADAS

Para comprender las ventajas de cada metodología se creó una tabla que contiene las características más importantes y estas se convertirán en la base para la construcción del procedimiento, para la aplicación móvil PQRDS de la Universidad de Pamplona.

Metodología	Destacado	Falencia
XP	1. Se debe realizar las historias de usuario, que describen la características que el producto a desarrollar tiene que poseer. 2. La comunicación, el ambiente dentro del equipo de desarrollo debe ser de colaboración y el equipo debe involucrar al cliente informándolo de las novedades en todo momento y viceversa por parte del cliente con el equipo. 3. El diseño debe ser lo más sencillo posible, dado que lo más relevante es la funcionalidad solicitada por el cliente. El código implementado, simplificarlo al máximo (refactorización). 4. Antes de empezar la implementación, el programador realiza pruebas dirigidas al funcionamiento de nuevas adiciones al sistema. 5. Desde el comienzo, los desarrolladores y el cliente deben tener la misma visión del proyecto a realizar. 6. Se deben realizar pequeñas entregas de módulos funcionales completos. 7. El código de los módulos debe ser integrado constantemente, el tiempo	1. Existe la necesidad de contar con la presencia de manera frecuente del cliente, que la mayoría de veces resulta complicado de lograr.

	de integración no debería pasar de un día. 8. Utilización de metáforas. 9. Debe aplicarse estándares de programación al código fuente. 10. No se debe trabajar horas extras.	
SCRUM	1. Sus tres pilares: Transparencia, los aspectos significativos del proceso deben ser conocido por todos los involucrados en el proyecto. Inspección, el progreso hacia un objetivo debe ser revisado de forma diligente por inspectores expertos para detectar variaciones. Adaptación, depende de la inspección, si un proceso se desvía de los límites aceptables debe ser ajustado inmediatamente. 2. El Sprint, el equipo de desarrollo se dedica sin interrupciones a realizar un avance del producto que puede ser potencialmente entregable. 3. Antes de cada Sprint se debe realizar una reunión para planificar dicho Sprint. 4. Establecer meta, que debe ser alcanzada al final de cada Sprint. 5. Se debe realizar el Scrum diario, para que el equipo de desarrollo sincronice sus actividades y cree un plan para el resto de la jornada laboral. 6. El grupo debe inspeccionarse a sí mismo y crear un plan de mejoras que sean tenidas en cuenta durante el siguiente Sprint. 7. Se debe realizar la lista de producto, es donde se lleva a cabo la	1. El tiempo para desarrollar o implementar está sujeto a una duración específica (30 días). 2. Se necesita de muchas semanas de prácticas de esta metodología para llegar a dominarla. 3. Se debe contar si o si con un Scrum Master para que guie al equipo de desarrollo en todo el proceso de Scrum.

	descripción del producto a desarrollar. 8. Se debe elaborar la respectiva lista de pendientes del Sprint, los elementos de la lista de producto seleccionados para el Sprint.	
KANBAN	1. Se debe visualizar el flujo de trabajo, se divide el trabajo a realizar en partes o tarjetas con su respectiva fecha de inicio, fecha de entrega y descripción de la actividad a desarrollar, todo esto con el objetivo de que los integrantes tengan claro el trabajo que van a realizar. 2. Se debe determinar el límite de trabajo en curso, el número de tareas en cada fase del ciclo tiene que ser establecido, para poder agregar una nueva tarea es necesario que otra se haya terminado. 3. Se debe medir el tiempo en completar una tarea, se mide desde que se realiza la petición hasta que termina la tarea para determinar el rendimiento del proceso. 4. No se utilizan roles.	1. Se necesita conocimiento de otra u otras metodologías de desarrollo ágil, gracias a que carece de algunas reglas esenciales para el desarrollo de software.
CRYSTAL	1. Se adapta a varios tipos de proyectos dependiendo de sus necesidades. 2. Se debe hacer menos énfasis en la documentación y más en versiones que corran y puedan ser ejecutadas. 3. El proceso de ejecución de la familia Crystal es menos disciplinado que otras metodologías debido a que intercambia productividad por facilidad de ejecución. 4. Se debe realizar una revisión al final de cada iteración con el objetivo de	1. No puede ser integrante del equipo un diseñador que no tenga conocimientos en programación. 2. El grupo de desarrollo utiliza tarjetas visuales para implementar en orden preferido, ignorando las relaciones de dependencia entre las tarjetas.

	buscar y corregir los posibles errores presentados durante el proceso de desarrollo. 5. Las personas involucradas en el proyecto se deben comunicar cara a cara, con su pregunta y su respectiva respuesta en el mismo espacio de tiempo.	3. Escasa documentación acerca del proceso para su aplicación.
MOBILE-D	1. Está basada en las metodologías ágiles de desarrollo XP y Crystal. 2. El desarrollo es basado en pruebas, se debe hacer integración continua de código, tareas de mejora de proceso de software y refactorización. 3. El límite de desarrolladores debe ser de 10 y el plazo máximo para desarrollar el producto, es de 10 semanas. 4. Para el desarrollo tiene sus fases de manera general muy bien establecidas (Exploración, Iniciación, Producción, Estabilización y Prueba del Sistema).	1. Carece de una guía detallada en sus fases, especialmente en la parte donde se planifica el proyecto a desarrollar. 2. El equipo de desarrollo necesita conocimiento de metodologías de desarrollo ágil, debido a que falta especificación en alguna de sus fases. 3. Demuestra similitud al modelo cascada, donde no específica retroceso a fases anteriores, dejando abierta la posibilidad de que el equipo de desarrollo (afectando a desarrolladores novatos) se ubique en cualquier fase anterior en el momento de realizar ajustes.

2.1.4 SISTEMA DE PETICIONES, QUEJAS, RECLAMOS, DENUNCIAS Y SUGERENCIAS

Es un conjunto de información organizada por una entidad con el objetivo de analizarla para brindar el mejor servicio posible a sus clientes, estos datos son obtenidos gracias al registro de solicitudes de los usuarios o clientes.

El sistema de PQRDS de la Universidad de Pamplona es una herramienta que le permite conocer o descubrir los aspectos que debe mejorar para ofrecer una mayor calidad en sus servicios.

El usuario de la entidad (en este caso la Universidad de Pamplona), cuenta con un espacio donde tiene la oportunidad de expresar sus inquietudes o inconformidades con el fin de recibir un mejor servicio.

2.1.5 NORMA ISO 9126

La evaluación de la calidad establecida en el procedimiento para realizar la prueba de la aplicación móvil es basada en la norma ISO 9126, que estable que "cualquier componente de la calidad del software puede ser descrito en términos de una o más de seis características básicas, las cuales son: funcionalidad, confiabilidad, usabilidad, eficiencia, mentenibilidad y portabilidad". (Abud Figueroa, 2004, p.1)

A continuación se describen cada una de las características mencionadas anteriormente:

Funcionalidad

En este grupo se conjunta una serie de atributos que permiten calificar si un producto de software maneja en forma adecuada el conjunto de funciones que satisfagan las necesidades para las cuales fue diseñado. Para este propósito se establecen los siguientes atributos: (Abud Figueroa, 2004, p.1)

- *Adecuación.* Se enfoca a evaluar si el software cuenta con un conjunto de funciones apropiadas para efectuar las tareas que fueron especificadas en su definición.
- *Exactitud.* Este atributo permite evaluar si el software presenta resultados o efectos acordes a las necesidades para las cuales fue creado.
- *Interoperabilidad.* Permite evaluar la habilidad del software de interactuar con otros sistemas previamente especificados.

- *Conformidad.* Evalúa si el software se adhiere a estándares, convenciones o regulaciones en leyes y prescripciones similares.
- *Seguridad.* Se refiere a la habilidad de prevenir el acceso no autorizado, ya sea accidental o premeditado, a los programas y datos.

Confiabilidad

Aquí se agrupan un conjunto de atributos que se refieren a la capacidad del software de mantener su nivel de ejecución bajo condiciones normales en un periodo de tiempo establecido. Las subcaracterísticas que el estándar sugiere son:

- *Nivel de Madurez*. Permite medir la frecuencia de falla por errores en el software.
- *Tolerancia a fallas*. Se refiere a la habilidad de mantener un nivel específico de funcionamiento en caso de fallas del software o de cometer infracciones de su interfaz específica.
- *Recuperación*. Se refiere a la capacidad de restablecer el nivel de operación y recobrar los datos que hayan sido afectados directamente por una falla, así como al tiempo y el esfuerzo necesarios para lograrlo.

Usabilidad

Consiste de un conjunto de atributos que permiten evaluar el esfuerzo necesario que deberá invertir el usuario para utilizar el sistema.

- *Comprensibilidad*. Se refiere al esfuerzo requerido por los usuarios para reconocer la estructura lógica del sistema y los conceptos relativos a la aplicación del software.
- *Facilidad de Aprender*. Establece atributos del software relativos al esfuerzo que los usuarios deben hacer para aprender a usar la aplicación.
- *Operabilidad*. Agrupa los conceptos que evalúan la operación y el control del sistema.

Eficiencia

Esta característica permite evaluar la relación entre el nivel de funcionamiento del software y la cantidad de recursos usados. Los aspectos a evaluar son:

- *Comportamiento con respecto al Tiempo*. Atributos del software relativos a los tiempos de respuesta y de procesamiento de los datos.
- *Comportamiento con respecto a Recursos*. Atributos del software relativos a la cantidad de recursos usados y la duración de su uso en la realización de sus funciones.

Mantenibilidad

Se refiere a los atributos que permiten medir el esfuerzo necesario para realizar modificaciones al software, ya sea por la corrección de errores o por el incremento de funcionalidad. En este caso, se tienen los siguientes factores:

- *Capacidad de análisis*. Relativo al esfuerzo necesario para diagnosticar las deficiencias o causas de fallas, o para identificar las partes que deberán ser modificadas.
- *Capacidad de modificación*. Mide el esfuerzo necesario para modificar aspectos del software, remover fallas o adaptar el software para que funcione en un ambiente diferente.
- *Estabilidad*. Permite evaluar los riesgos de efectos inesperados debidos a las modificaciones realizadas al software.
- *Facilidad de Prueba*. Se refiere al esfuerzo necesario para validar el software una vez que fue modificado.

Portabilidad

En este caso, se refiere a la habilidad del software de ser transferido de un ambiente a otro, y considera los siguientes aspectos:

- *Adaptabilidad*. Evalúa la oportunidad para adaptar el software a diferentes ambientes sin necesidad de aplicarle modificaciones.

- *Facilidad de Instalación*. Es el esfuerzo necesario para instalar el software en un ambiente determinado.
- *Conformidad*. Permite evaluar si el software se adhiere a estándares o convenciones relativas a portabilidad.
- *Capacidad de reemplazo*. Se refiere a la oportunidad y el esfuerzo usado en sustituir el software por otro producto con funciones similares.

2.2 ESTADO DEL ARTE

Este apartado contiene trabajos realizados, los cuales están relacionados con este proyecto. Trabajos que se han realizado a nivel nacional e internacional.

Nivel Nacional:

Año	Tema	Características
2013	Metodologías ágiles en el desarrollo de aplicaciones para dispositivos móviles. Estado actual	En este trabajo, el autor (Yohn Amaya) se dedicó a realizar una serie de síntesis sobre las metodologías de desarrollo de software con mayor presencia de documentación en internet y con equipos de tamaño reducido (XP, Scrum y Test Driven Development), sobre el desarrollo de aplicaciones para dispositivos móviles, los sistemas operativos (Android, iOS, mobile web, Windows pone, etc.) más populares para dispositivos móviles y sobre las metodologías de desarrollo usadas (en ese instante de tiempo) para la implementación de aplicaciones para dichos dispositivos.
2013	Metodología para el desarrollo de aplicaciones móviles	En este trabajo, los autores (Maira Gasca & otros) generaron una propuesta metodológica basada en experiencia de investigaciones previas en el contexto de las aplicaciones móviles, en los conceptos de Ingeniería de software

		educativo con modelado orientado a objetos y valores de las metodologías ágiles de desarrollo de software para la implementación de aplicaciones móviles, propuesta metodológica que fue probada con la creación una aplicación para dispositivos móviles que llamaron Dr Móvil.

Del tema "Metodologías ágiles en el desarrollo de aplicaciones para dispositivos móviles. Estado actual" resalto el énfasis que el autor hace hacia las metodologías ágiles como la mejor alternativa para desarrollar aplicaciones móviles, donde destaca que la metodología ágil a ser utilizada debe contener una serie de ajustes debido a las condiciones especiales que presentan los dispositivos móviles.

Del tema "Metodología para el desarrollo de aplicaciones móviles" es interesante la forma como los autores de este proyecto generaron la estructura de su metodología, organizada en gran parte de acuerdo a su fundamentación teórica. También es de destacar las características que deben tener las aplicaciones en el contexto móvil descritas en este proyecto.

Nivel Internacional:

Año	Tema	Características
2009	Metodología de desarrollo ágil para sistemas móviles. Introducción al desarrollo con Android y el iPhone	Los autores (Paco Blanco & otros) describieron una antesala hacia el desarrollo de aplicaciones para dispositivos móviles, destacando que el uso de las metodologías ágiles es el medio más apropiado para dichos desarrollos, pero resaltando que las metodologías ágiles en su momento deberían tener algunas adaptaciones o ajustes debido a las condiciones especiales que presentan los dispositivos móviles. También destacaron lo que podrían ser las características básicas

		(agilidad, conciencia de mercado, soporte a la línea de producción de software, desarrollo basado en arquitectura, soporte para reusabilidad, inclusión de sesiones de revisión y de aprendizaje, y especificación temprana de la arquitectura física) de una metodología ideal para el proceso de desarrollo de software para dispositivos móviles. Y describen una metodología (Mobile-D) orientada hacia la construcción de aplicaciones para dispositivos móviles.

Del tema "Metodología de desarrollo ágil para sistemas móviles. Introducción al desarrollo con Android y el iPhone" es de resaltar los aportes por parte de los autores sobre las características básicas que debe tener una metodología de desarrollo de software para ser ideal en el proceso de desarrollo de aplicaciones para plataformas móviles.

CAPÍTULO 3. METODOLOGÍA DE INVESTIGACIÓN

Este capítulo contiene el conjunto de pasos, los cuales son de vital importancia ya que permiten desarrollar de manera organizada el proceso de investigación. Los siguientes pasos son basados en (Hernández Sampieri, Fernández Collado, & Baptista Lucio, s.f).

3.1 PARADIGMA

Este proyecto de investigación tiene un enfoque cualitativo ya que la recolección de datos es realizada sin medición numérica (uso de fórmulas estadísticas), sin algún tipo de predicciones seguras. La recolección de datos fue surgiendo durante la investigación, dependiendo de la importancia que cobraba el análisis o el procesamiento de la información obtenida.

3.2 ALCANCE

Este proyecto de investigación tiene un alcance exploratorio ya que en la Universidad de Pamplona no existe un procedimiento establecido para la implementación de aplicaciones móviles con características cliente-servidor, que en este caso ha servido para la creación de la aplicación PQRDS de la Universidad de Pamplona para dispositivos móviles.

3.3 TIPO

Este proyecto de investigación es de tipo no experimental ya que no se hace uso de experimento alguno.

3.4 FUENTES DE INFORMACIÓN

Fue oportuno investigar en fuentes de cualidad virtual, donde se hizo uso de archivos PDF y páginas web, todas estas fuentes sumamente confiables (la mayoría artículos aceptados y libros populares); oportuno ya que contamos con los servicios que nos presta la tecnología los cuales nos hacen la vida más fácil disminuyendo el tiempo empleado en cuanto a la búsqueda de la información necesaria se refiere. Dentro de las

fuentes de información se contó con el gestor de búsqueda google académico, también participaron los Ingenieros que laboran como analistas de desarrollo en la oficina de Desarrollo Específico ubicada en el CIADTI.

3.5 POBLACIÓN

Este proyecto está dirigido inicialmente a toda la comunidad que se vea afectada por el entorno de la Universidad de Pamplona, ofreciendo un nuevo canal de comunicación para que puedan expresar sus inquietudes, sugerencias, etc. Y por otro lado dirigido a los estudiantes de pregrado del programa de Ingeniería de Sistemas de la Universidad de Pamplona, que quieran implementar aplicaciones para dispositivos móviles con características cliente-servidor, gracias a uno de los resultados de este proyecto, que es el procedimiento elaborado para la realización dicha aplicación móvil.

3.6 INSTRUMENTOS

Como instrumento de recolección de información, se ha tomado como principal herramienta la observación y el seguimiento a los procesos que se llevan a cabo en la oficina de Desarrollo Específico en el CIDATI, donde se obtenía información redactándose en hojas gracias a conversaciones con los Ingenieros que allí laboran.

3.7 CONSIDERACIONES ÉTICAS

El resultado principal de este proyecto (aplicación móvil para el sistema PQRDS de la Universidad de Pamplona), es propiedad de la Universidad de Pamplona, está orientado hacia toda la comunidad en general, sin distinción de clase alguna (clase social, cultura, creencias religiosas, etc.). Y el resultado secundario (procedimiento para la construcción de aplicaciones móviles cliente-servidor) es libre de ser utilizado por la persona que así lo desee.

CAPÍTULO 4. PROPUESTA DEL PROCEDIMIENTO Y RESULTADOS

Contiene el producto final de este proyecto; este capítulo se divide en dos partes, el procedimiento para realizar la aplicación móvil del sistema de PQRDS de la Universidad de Pamplona y la aplicación móvil del sistema de PQRDS implementada a través del procedimiento.

A continuación se presenta el procedimiento que tiene como nombre **AppSIERRA** y las ilustraciones de la implementación de la aplicación móvil a través del procedimiento.

4.1 PROCEDIMIENTO

El siguiente procedimiento está diseñado para el desarrollo de aplicaciones móviles, está constituido por una estructura basada en las fases de desarrollo propuestas en la metodología para el desarrollo de aplicaciones móviles Mobile-D. Las partes que componen esta estructura son tomadas de las metodologías ágiles de desarrollo XP, Scrum, Kanban, la familia de metodologías Crystal y por experiencia propia en desarrollo de app.

AppSIERRA

Es una guía diseñada para desarrollar aplicaciones para dispositivos móviles, con características *cliente – servidor*, consta de cinco Etapas (ver *Figura 4.1*).

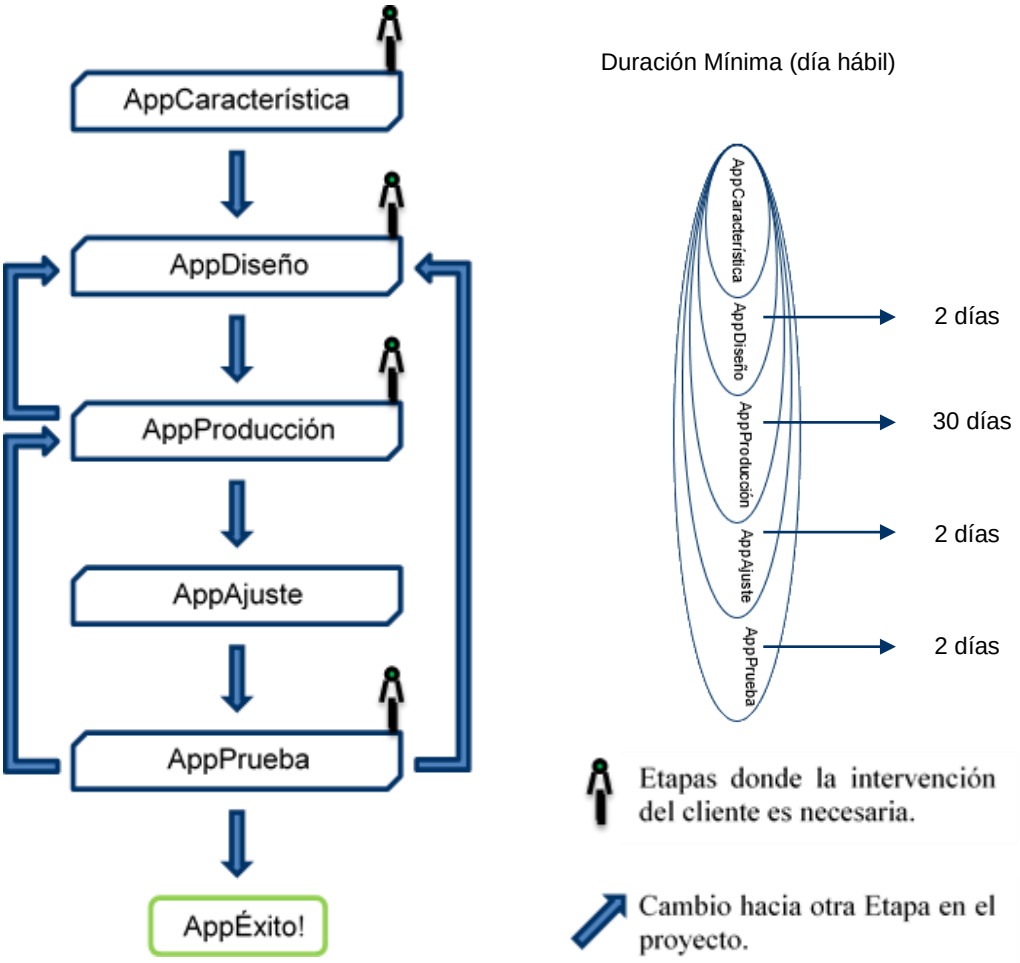


Figura 4.1. Proceso AppSIERRA.

El desarrollo de una aplicación móvil bajo los pasos propuestos por este procedimiento, debería demorar al menos 36 días hábiles.

4.1.1 Etapa 1. AppCaracterística

Los requerimientos para desarrollar la aplicación móvil se obtienen Interactuando con el cliente del proyecto, que debe describir con sus propias palabras cuales son las características que desea para su producto.

Estas características se deben redactar inmediatamente en el Formato 1 que se ilustra a continuación, después de la descripción realizada por el cliente. El desarrollador después de atender los requerimientos del cliente, procede a establecer las características que está dispuesto a implementar, luego debe existir un diálogo cliente-desarrollador para llegar a un acuerdo. Una vez llegado a común acuerdo sobre los requerimientos del producto se procede a la firma de ambas partes (desarrollador y cliente) para oficializar el cumplimiento de la Etapa 1.

Para definir la fecha de entrega de la aplicación, se establece segmentando las entregas de la aplicación con relación a las Etapas, se van definiendo tiempos de entregas en las diferentes Etapas. Las fechas se deben diligenciar en el Formato 2.

Formato 1. CARACTERÍSTICAS APLICACIÓN MÓVIL

Nombre de la Aplicación	
Dueño de Producto: **Ced o NIT:** **Desarrollador(es):** **Ced:**	
Características ofrecidas para la aplicación	
Cliente	**Desarrollador**
Características acordadas para la aplicación	
Valor ($):	**Fecha:**

_____ _____

Dueño de Producto Desarrollador

Formato 2. PROGRAMACIÓN ENTREGA AVANCE

Nombre de la Aplicación:	
Nombre de la Etapa: **Iteración N°**	
Fecha:	**Fecha de Entrega:**

Desarrollador

Nombre de la Aplicación:	
Nombre de la Etapa: **Iteración N°**	
Fecha:	**Fecha de Entrega:**

Dueño de Producto

4.1.2 Etapa 2. AppDiseño

En el momento de iniciar el diseño de la aplicación móvil se debe llevar por el camino más sencillo o simple que se pueda, el tiempo para realizar el diseño no puede sobrepasar 2 días hábiles, una vez finalizado el diseño, someterlo a revisión con el cliente, a partir de una prueba de aceptación (ver *Figura 4.2*).

Si es aprobado, se avanza a la etapa AppProducción donde se establece la fecha de entrega del diseño de la aplicación móvil funcionando correctamente. Si es reprobado, ajustar el diseño inmediatamente hasta obtener el "aprobado".

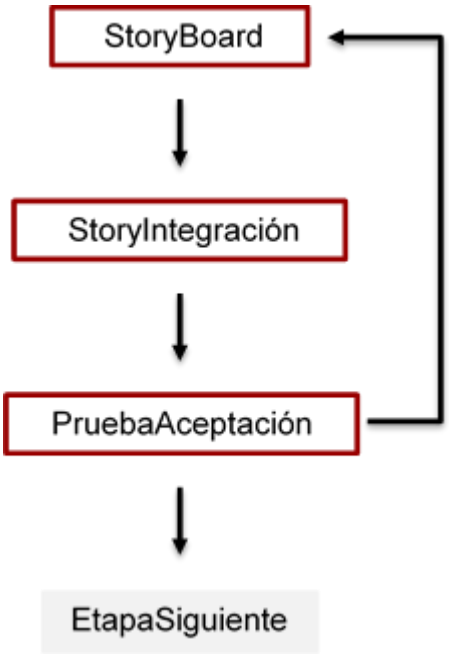


Figura 4.2. Etapa AppDiseño.

- StoryBoard:
  1. Tomar papel y lápiz, realizar el StoryBoard para describir de manera gráfica cada una de las vistas de la aplicación y asignarle un título a cada dibujo realizado (deje un espacio para redactar las características del dibujo). Se recomienda tomar hojas de cuaderno pequeño, de manera horizontal dividirla por medio de líneas en 3 partes y en cada parte realizar un dibujo, o si utiliza hojas de tamaño carta, dividirla en 6 partes iguales.
  2. Redactar las características pertenecientes a cada interfaz gráfica, preferiblemente debajo de cada dibujo (interfaz).

- StoryBoardIntegración:
  1. Integrar o concatenar los dibujos realizados por medio de flechas o apuntadores. Si nota que puede conectar de manera organizada los dibujos dentro de la misma hoja, intégrelos inmediatamente. De lo contrario separe las partes de la hoja y adjúntelas en una cartulina (o algún material parecido donde pueda presentar todo el esquema al cliente). También puede dibujar el esquema en una nueva hoja solo con los títulos de cada interfaz.
  2. Para la integración de las partes, realice el esquema de acuerdo a las dependencias de las funcionalidades, es decir, la interfaz gráfica1 que al realizar una determinada acción abre una interfaz gráfica2, asignarle un apuntador de la gráfica1 apuntando a la gráfica2 denotando una relación de dependencia. Una vez integrado todos los dibujos del StoryBoard, ya está listo el diseño para ser presentado al cliente.

- PruebaAceptación:
  1. Presentar el esquema al cliente, atendiendo pacientemente las dudas que se le puedan presentar, apoyado en el formato generado en la etapa AppCaracterística.
  2. Si el cliente acepta el esquema, el diseño está en Etapa completada de manera temporal, ya puede avanzar a la siguiente Etapa. Si no lo acepta, tomar pacientemente las observaciones o correcciones y volver a la fase *StoryBoard* para aplicar las correcciones.

4.1.3 Etapa 3. AppProducción

Esta etapa se divide en dos fases, la fase de producción de vistas y la fase de producción de servicios (ver *Figura 4.3*).

Figura 4.3. Etapa AppProducción.

En la fase *Vistas* se implementa toda la interfaz gráfica diseñada en la Etapa AppDiseño, esta implementación se realiza en un Sprint de máximo 10 días hábiles.

En la fase *Servicios* se implementa las funcionalidades de cada una de las vistas implementadas en la fase *Vistas*, esta implementación se realiza en un Sprint de máximo 20 días hábiles.

A continuación se describen los detalles de cada una de estas fases:

Fase Vistas

Antes de iniciar la fase Vistas, organizar un tablero con las diferentes imágenes creadas en el storyBoard con una secuencia lógica de avance, identificando el estado de desarrollo a nivel de vistas, si es pendiente, desarrollo, prueba o terminado. La estructura de esta fase se ilustra a continuación (*Figura 4.4*):

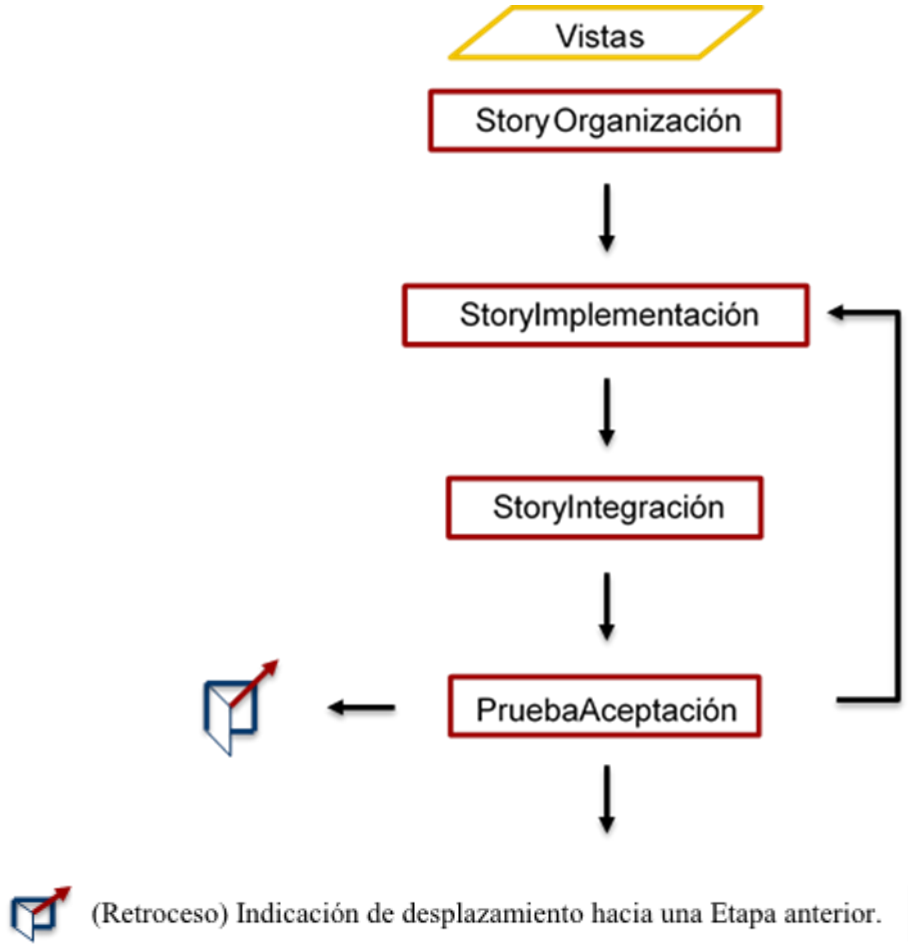


Figura 4.4. Fase Vistas.

- StoryOrganización:
  1. Tomar un tablero, dividirlo en 4 columnas (pendiente, desarrollo (2), prueba (2) y terminado).
  2. Todos los bloques (que son los títulos de cada imagen del StoryBoard que fueron previamente diseñados) antes de comenzar la implementación, van a la columna "pendiente".

- StoryImplementación:
  1. Cada bloque a implementar va a la columna "desarrollo", va un solo bloque a dicha columna, en caso de que se presente problemas o retrasos en la implementación de un bloque, asignar otro a la columna "desarrollo", pero no asigne más de dos bloques a dicha columna; de ahí la razón de ser del número dos (2) que acompaña al título de la columna (desarrollo).
  2. Cada bloque implementado pasa a la columna "prueba", donde se revisa que todo esté en orden con el bloque implementado (funcionando correctamente), sin pasar el límite de asignaciones (2).
  3. Cada bloque que supere la prueba exitosamente, pasa a la columna "terminado".

- StoryIntegración:
  1. Cuando todos los bloques estén en la columna "Terminado", integrar todas las vistas implementadas de la aplicación móvil en el mismo orden de secuencia que se integró el StoryBoard en la Etapa de diseño.

- PruebaAceptación:
  1. Presentar el esquema implementado (interfaz gráfica de la aplicación móvil) al cliente, atendiendo atentamente las dudas que se le puedan presentar.
  2. Si el cliente acepta la interfaz gráfica de la app, la implementación del diseño está en etapa completada de manera temporal, y ya puede iniciar la implementación de los *servicios*. Si no lo acepta, tomar pacientemente las observaciones o correcciones y volver a la Etapa **AppDiseño** para ajustar el diseño aplicando las correcciones.

Nota: Si calcula que al término de dos semanas no va a completar la StoryImplementación, realizar una StoryIntegración temporal, presentar al cliente el avance obtenido y programar un nuevo Sprint.

Fase Servicios

En esta fase se implementa el funcionamiento de cada una de las vistas de la aplicación móvil (ver *Figura 4.5*):

Figura 4.5. Fase Servicios.

Para realizar la implementación de los servicios ofrecidos por cada una de las interfaces gráficas de la aplicación móvil, los pasos son parecidos a los desarrollados anteriormente para implementar las *vistas*, pero esta vez orientados a la función o funciones que ofrece cada vista en el Smartphone.

- StoryOrganización:
  1. Tomar el tablero, con sus respectivas 4 columnas (pendiente, desarrollo (2), prueba (2) y terminado).
  2. Asignar los bloques a implementar a la columna "pendiente".

- StoryImplementación:
  1. Cada bloque a implementar va a la columna "desarrollo", va un solo bloque a dicha columna, en caso de que se presente problemas o retrasos en la implementación del servicio de un bloque, asignar otro a la columna "desarrollo", sin pasar el límite de asignaciones (2).
  2. Cada bloque implementado pasa a la columna "prueba", donde se revisa que todo esté en orden con el servicio implementado, sin pasar el límite de asignaciones (2).
  3. Cada bloque que supere la prueba exitosamente, pasa a la columna "terminado".

Cada ServicioCompletado es un ServicioIntegrado automáticamente, debido a que la interfaz gráfica de la aplicación móvil ya se encuentra integrada. Una vez completado el último servicio, ya puede avanzar a la siguiente Etapa.

Nota: Si calcula que al término de un mes no va a completar la StoryImplementación, programar un nuevo Sprint notificando al cliente.

4.1.4 Etapa 4. AppAjuste

Ultimar los detalles de la interfaz gráfica con el objetivo de darle un aspecto agradable a la aplicación móvil no está de más. Los ajustes se deberían terminar en un tiempo máximo de dos días hábiles (ver *Figura 4.6*).

Refactorizar el código producido para su fácil legibilidad y mantenimiento, y por qué no para lograr una mayor eficiencia en la ejecución de la aplicación.

Una vez terminado todos los detalles de la app, la Etapa 4 ha sido temporalmente completada.

Figura 4.6. Etapa AppAjuste.

- AspectoElegante:
  1. Asignarle estilos a la interfaz gráfica de la aplicación móvil, por ejemplo, colores (los colores empleados que se combinen perfectamente) y bordes. Como sugerencia, tener en cuenta colores que no cansen la vista del usuario, para más información consulte la teoría del color.
  2. Los iconos utilizados deben ser vistosos para el usuario y coherentes con la funcionalidad ofrecida. Como sugerencia, tener en cuenta que el tamaño de los iconos ocupe el tamaño abarcado por la huella dactilar (evitar esfuerzos por parte del usuario al momento de presionar dicho icono) y que las imágenes utilizadas para los iconos tengan una buena resolución, pero a la vez no tengan un peso elevado, pues se puede tornar un poco lento al momento de cargar la interfaz gráfica de la aplicación.

- Refactorización:
  1. Simplificar el código producido, por ejemplo, las líneas de código repetidas eliminarlas utilizando un método que agrupe dicho código. Buscar bajo su criterio que el código quede fácilmente legible y de fácil mantenimiento.
  2. Una vez considerado que ha terminado la refactorización, ya puede avanzar a la siguiente Etapa.

4.1.5 Etapa 5. AppPrueba

Someter la aplicación móvil a una serie de revisiones, con el objetivo de descubrir posibles fallas y ofrecer un producto de calidad para satisfacer las necesidades del cliente (ver *Figura 4.7*), las revisiones o pruebas no deberían pasar de los 2 días hábiles. Una vez el cliente acepte la app, la Etapa 5 se ha completado y el proyecto finalizado.

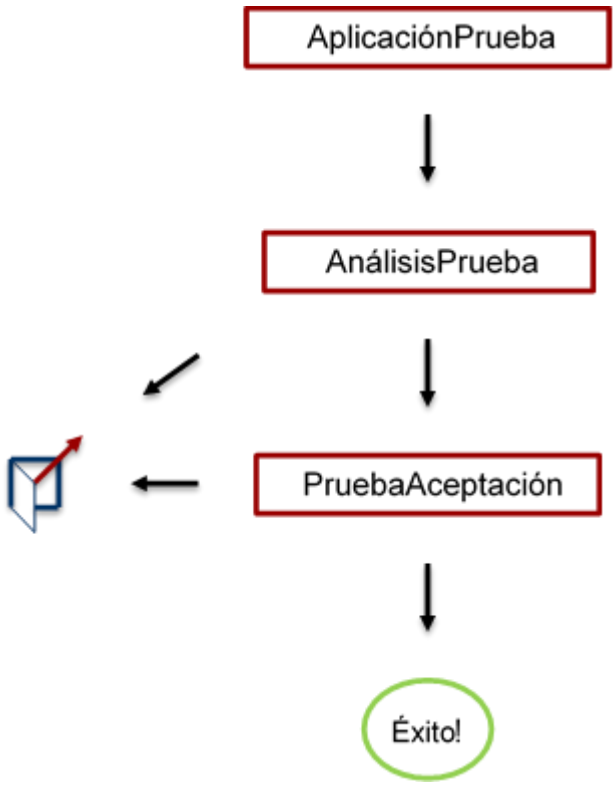


Figura 4.7. Etapa AppPrueba.

- AplicaciónPrueba:
  1. Diseñar un guion de pruebas, donde se valide el cumplimiento de las funcionalidades solicitadas por el cliente y las funcionalidades técnicas que esta aplicación debe cumplir, dicho guion debe enmarcar los pasos que siguen a continuación.
  2. Revisar si la aplicación móvil cumple con las funciones o características que se supone debe tener para satisfacer las necesidades del cliente.
  3. Utilizar todos los servicios de la app durante un periodo de tiempo prolongado de tal manera que esta se ejecute normalmente (sin fallas o errores).
  4. Medir la facilidad de uso, esto con ayuda de terceros que deben poder usar las funciones de la app intuitivamente, sin recibir ayuda alguna.
  5. Observar si los recursos de los Smartphone consumidos por la app son los necesarios. Los tiempos de respuesta deben ser lo más bajo posible, tener en cuenta que entre menos recursos se consuman y más rápido sea la ejecución, mayor será la satisfacción del usuario.
  6. Probar que la app funciona en diferentes tipos de Smartphone.

- AnálisisPrueba:
  1. Marcar cada uno de los objetivos cumplidos diseñados en el guion de pruebas, a través de los pasos restantes en la AplicaciónPrueba, los que falten por marcar ajustarlos volviendo a la Etapa correspondiente.
  2. Realizar un inventario con los elementos completados y los faltantes, en caso que exista fecha de reunión con el cliente para poder exponer y justificar el estado actual del proyecto.

- PruebaAceptación:
  1. Presentar la aplicación móvil funcionando correctamente al cliente, atendiendo pacientemente las dudas que se le puedan presentar.
  2. Si el cliente acepta la aplicación móvil, ya se encuentra en el estado **AppÉxito!** y todas las etapas anteriores se encuentran completadas de forma automática, dando por finalizado el proyecto de la aplicación móvil. Si el cliente reprueba la aplicación móvil, tomar pacientemente las observaciones o correcciones y

volver a la Etapa **AppDiseño** o **AppProducción**, según lo que necesite modificar para realizar los ajustes aplicando las correcciones.

Formato 3. FORMATO DE ACEPTACIÓN

Nombre Aplicación: _____

Fecha: _____ **Número:** ____

Errores	Mejoras

_____
Firma Aceptación

Recomendaciones.

El anterior procedimiento está diseñado para que la aplicación móvil sea implementada por un desarrollador, si van a trabajar dos o más personas, trate de adaptar el procedimiento, específicamente en la Etapa 3, modificando el límite de la columna "desarrollo" y "prueba", y repartiendo cargas de trabajo bajo sus criterios equitativos; si el presente procedimiento no se puede ajustar a su proyecto, la recomendación es buscar una metodología de desarrollo de software que mejor se adapte al proyecto a realizar.

No invertir demasiado tiempo en documentación, pues es valioso para terminar el producto lo más pronto posible y no hay que preocuparse por roles, solo existe el cliente y el o los desarrolladores.

En la producción de líneas de código, asignar etiquetas o identificadores por medio de comentarios, con el objetivo de proporcionar fácil legibilidad y mantenimiento; también es recomendable aplicar estándares de programación al código fuente.

A la menor duda presentada consultar con el cliente, para resolverla lo más pronto posible. Siempre debe existir disponibilidad para atender las intervenciones del cliente del proyecto exceptuando los Sprints.

4.2 APLICACIÓN MÓVIL PQRDS

La aplicación móvil es el resultado del presente trabajo de grado realizado bajo la modalidad de práctica empresarial en el CIADTI donde surge la necesidad de desarrollar el sistema de PQRDS de la Universidad de Pamplona para dispositivos móviles.

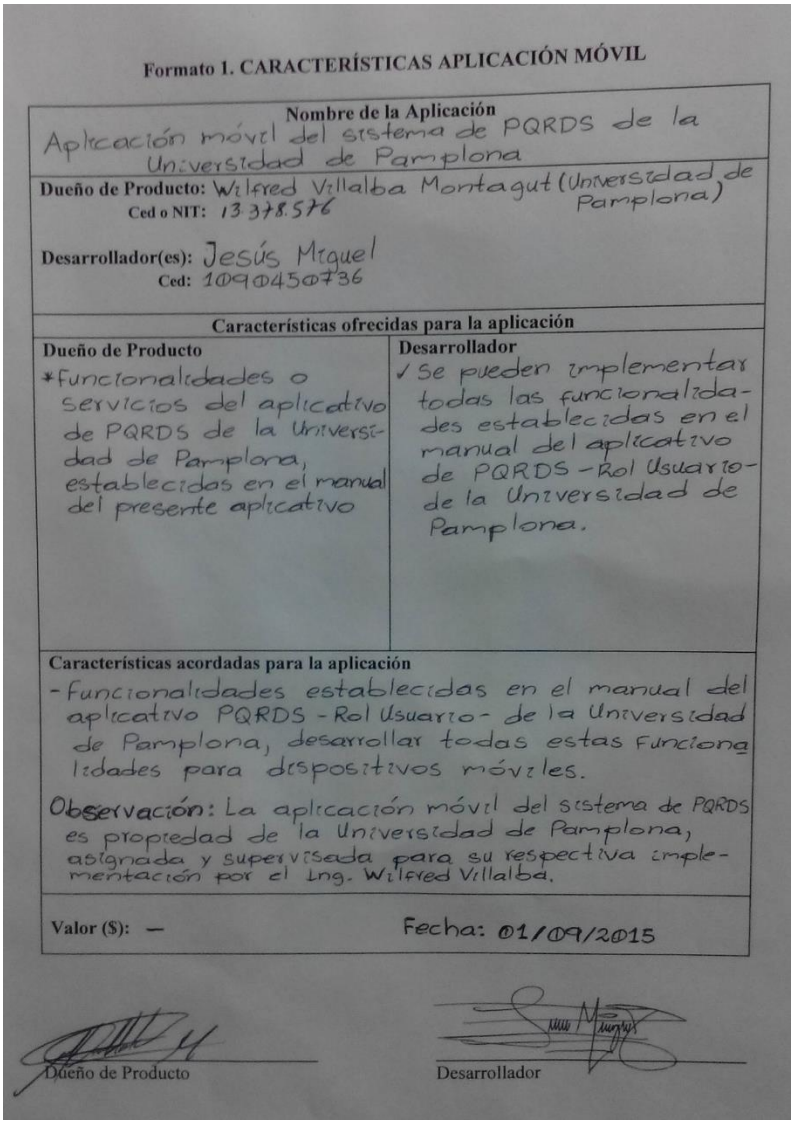
La aplicación se desarrolló y se implementó con la ayuda del procedimiento anteriormente descrito. A continuación se describe cada una de las Etapas por las que fue avanzando la construcción e implementación del sistema de PQRDS de la Universidad de Pamplona para dispositivos móviles, en este caso para el sistema operativo Android.

4.2.1 Etapa AppCaracterísticas

Debido a que la aplicación móvil del sistema de PQRDS de la Universidad de Pamplona debe ofrecer las mismas funcionalidades que la aplicación de PQRDS (para el rol usuario) de la Universidad que ya se encuentra construida para equipos de escritorios, las características para esta aplicación móvil surgen del manual de la aplicación para escritorios que ya está implementada, manual que fue suministrado por el coordinador técnico de desarrollo Wilfred Villalba Montagut de la oficina de Desarrollo Específico ubicada en el CIADTI.

El trabajo del desarrollador en esta Etapa fue leer y analizar los servicios o funcionalidades del manual que estaban al alcance para su implementación, donde el desarrollador después de un estudio de cada una de las funcionalidades aceptó el reto de implementar cada una de ellas.

No fue necesario establecer de manera formal las fechas de entregas para las Etapas correspondientes debido a que el desarrollador de esta aplicación se encontraba laborando en modalidad pasantía en el CIADTI bajo la supervisión del coordinador de la oficina Desarrollo Específico, que en este caso era el cliente o dueño de producto (ver Figura 4.8) asignado por parte de la Universidad.



Formato 1. CARACTERÍSTICAS APLICACIÓN MÓVIL

Nombre de la Aplicación
Aplicación móvil del sistema de PQRDS de la Universidad de Pamplona

Dueño de Producto: Wilfred Villalba Montagut (Universidad de Pamplona)
Ced o NIT: 13.378.576

Desarrollador(es): Jesús Miguel
Ced: 1090450736

Características ofrecidas para la aplicación

Dueño de Producto	Desarrollador
*Funcionalidades o servicios del aplicativo de PQRDS de la Universidad de Pamplona, establecidas en el manual del presente aplicativo	✓ Se pueden implementar todas las funcionalidades establecidas en el manual del aplicativo de PQRDS - Rol Usuario - de la Universidad de Pamplona.

Características acordadas para la aplicación
- Funcionalidades establecidas en el manual del aplicativo PQRDS - Rol Usuario - de la Universidad de Pamplona, desarrollar todas estas Funcionalidades para dispositivos móviles.

Observación: La aplicación móvil del sistema de PQRDS es propiedad de la Universidad de Pamplona, asignada y supervisada para su respectiva implementación por el Ing. Wilfred Villalba.

Valor ($): —
Fecha: 01/09/2015

Dueño de Producto
Desarrollador

Figura 4.8

4.2.2 Etapa AppDiseño

En esta Etapa se ha elaborado todo el diseño de la aplicación móvil que se ilustra a continuación:

StoryBoard

Los respectivos dibujos que componen el StoryBoard tienen cada uno su título (Inicio, Menú Principal, Registrar Requerimiento, Buscar Requerimiento, Listar Requerimiento, Ver Requerimiento, Actualizar Contraseña, Ver Información, Registro como Usuario, Recordar Usuario y Contraseña y Buscar Requerimiento no en Línea) en la parte superior, su estructura en el centro y sus correspondientes características en la parte inferior (ver Figura 4.9 – Figura 4.12). La última interfaz (Buscar Requerimientos no en Línea) no tiene asignado su dibujo ya que es exactamente igual a la interfaz "Ver Requerimiento".

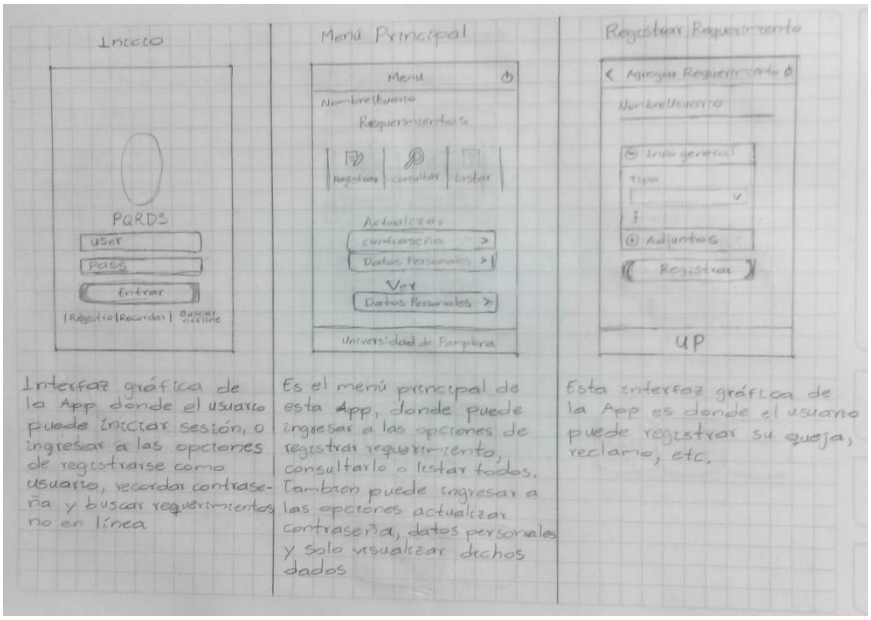


Figura 4.9

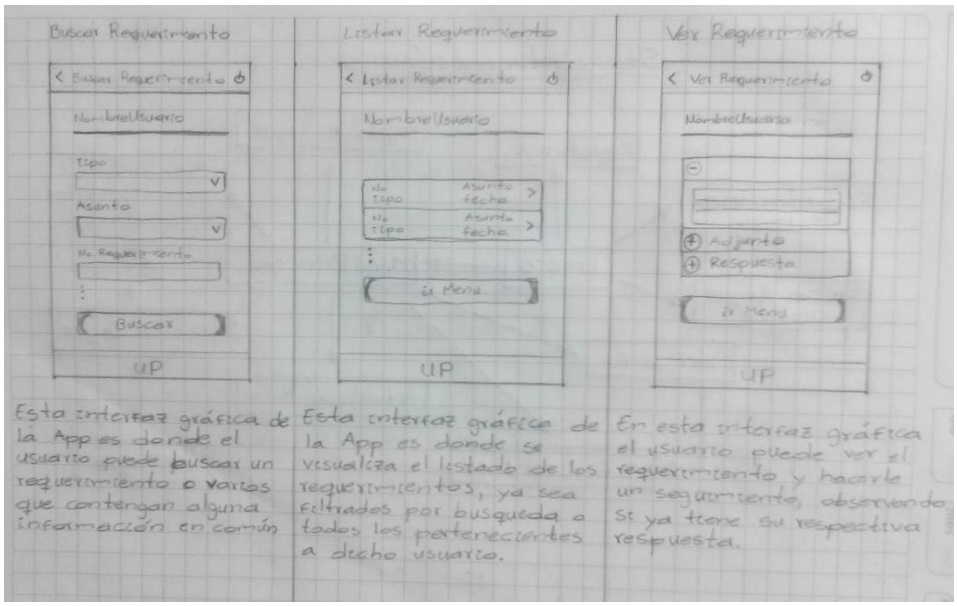


Figura 4.10

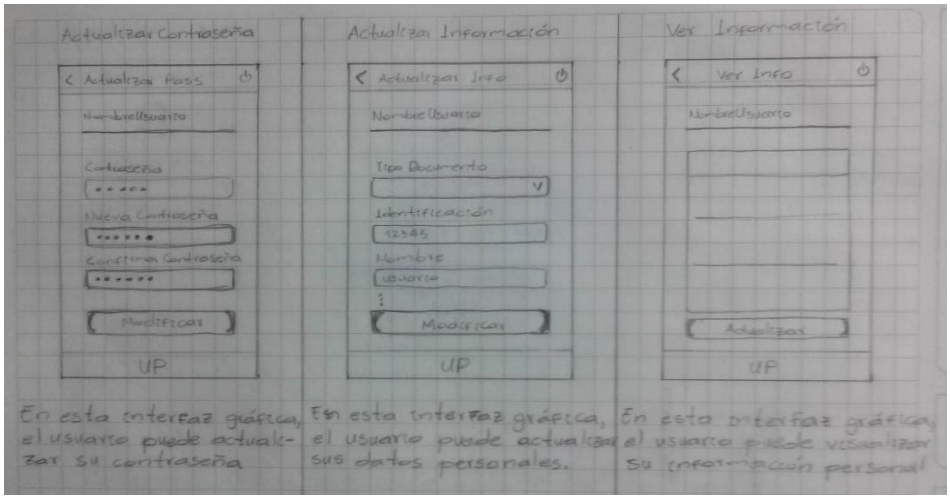


Figura 4.11

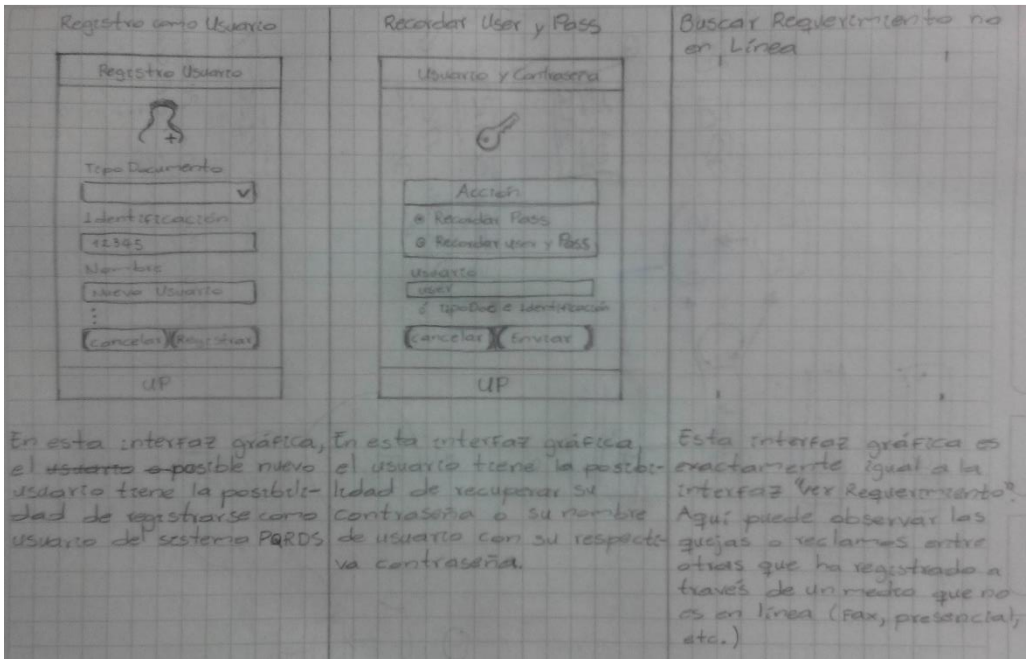

Registro como Usuario
Registro Usuario
Tipo Documento
Identificación
12345
Nombre
Nuevo Usuario
Cancelar
Registrar
UP
En esta interfaz gráfica, el ~~usuario~~ posible nuevo usuario tiene la posibilidad de registrarse como usuario del ssteria PQRDS
Recordar User y Pass
Usuario y Contraseña
Acción
Recordar Pass
Recordar user y Pass
usuario
user
tipoDoc e identificación
Cancelar
Enviar
UP
En esta interfaz gráfica el usuario tiene la posibilidad de recuperar su contraseña o su nombre de usuario con su respectiva contraseña.
Buscar Requerimiento no en Línea
Esta interfaz gráfica es exactamente igual a la interfaz "ver Requerimiento" Aquí puede observar las quejas o reclamos entre otras que ha registrado a través de un medio que no es en línea (Fax, presencial, etc.)

Figura 4.12

StoryBoardIntegración

Los dibujos se han integrado por medio de los títulos para mayor comodidad y ahorro de papel (ver Figura 4.13).

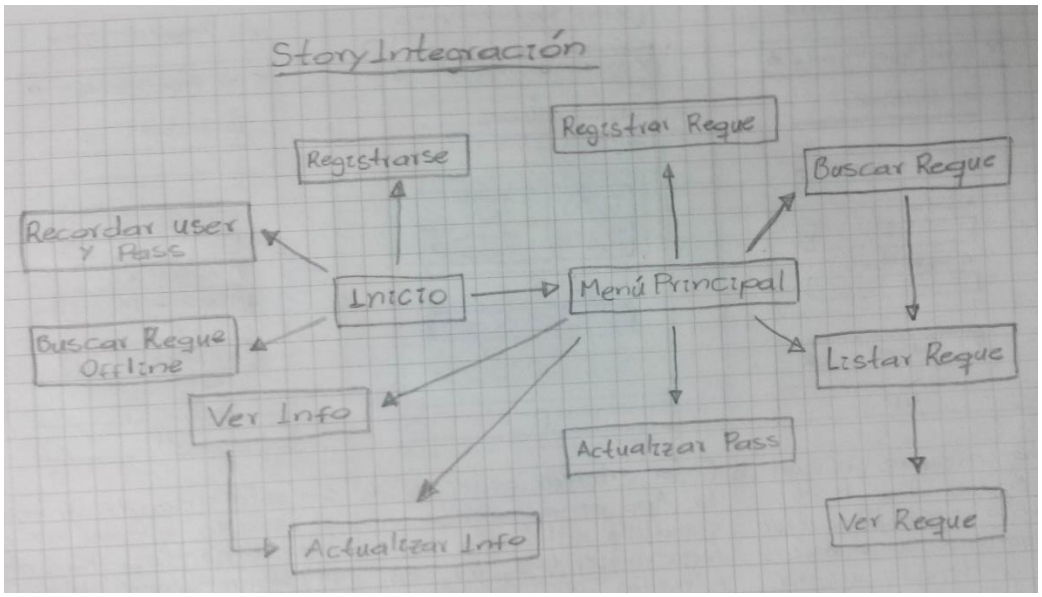


Figura 4.13

PruebaAceptación

El dueño de producto o cliente entendió perfectamente el StoryBoard apoyado en la StoryBoardIntegración, pero solicitó simplificar las interfaces "Inicio" y "Menú Principal" debido a que le notaba demasiada información. El ajuste se hizo inmediatamente en su presencia, obteniéndose como mejora las siguientes interfaces (ver Figura 4.14).

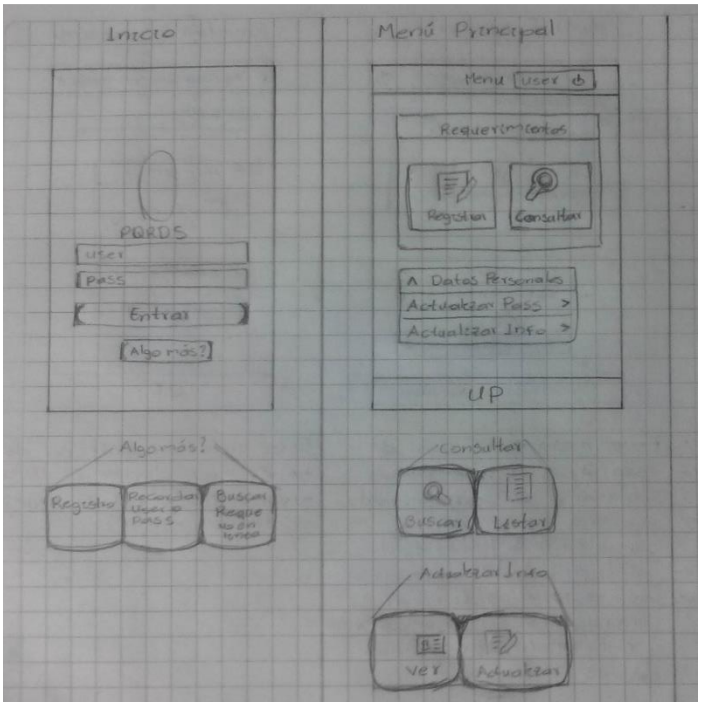


Figura 4.14

4.2.3 Etapa AppProducción

Por motivos de confidencialidad, solo se ilustra la interfaz gráfica implementada, omitiéndose todos los detalles del código fuente perteneciente a esta aplicación móvil.

Producción de Vistas:

StoryOrganización

A continuación (ver Figura 4.15 – Figura 4.16) se ilustra la organización de las partes o bloques justo antes de implementar las vistas de la aplicación.

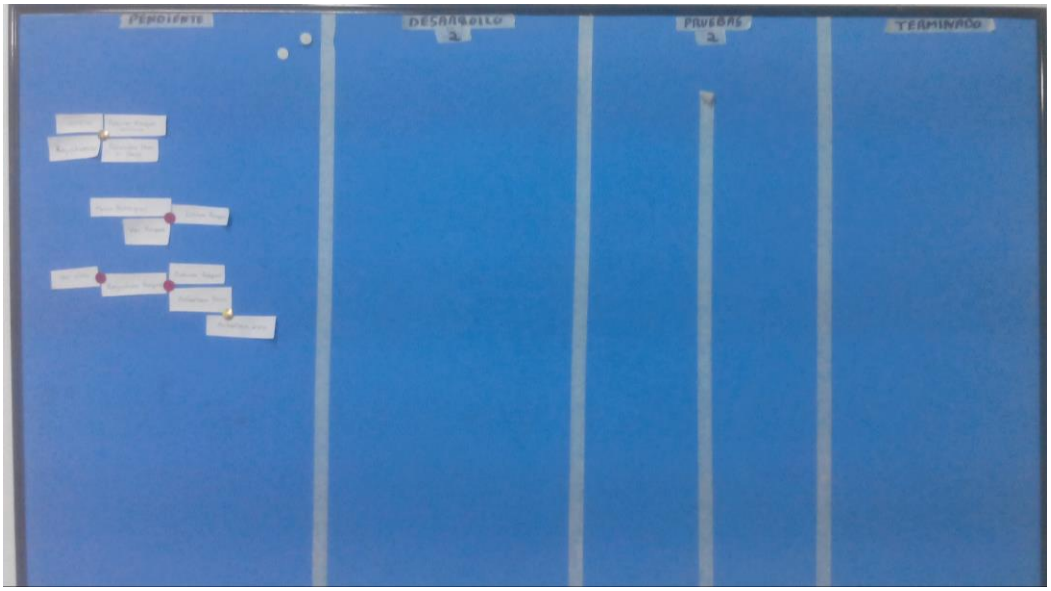


Figura 4.15

Los títulos que representan cada dibujo perteneciente al StoryBoard son los siguientes: Inicio, Buscar Requerimiento no en Línea, Registrarse como Usuario, Recordar Usuario y Contraseña, Menú Principal, Listar Requerimientos, Ver Requerimientos, Buscar Requerimientos, Registrar Requerimientos, Ver Datos Personales, Actualizar Datos Personales y por último Actualizar Contraseña.

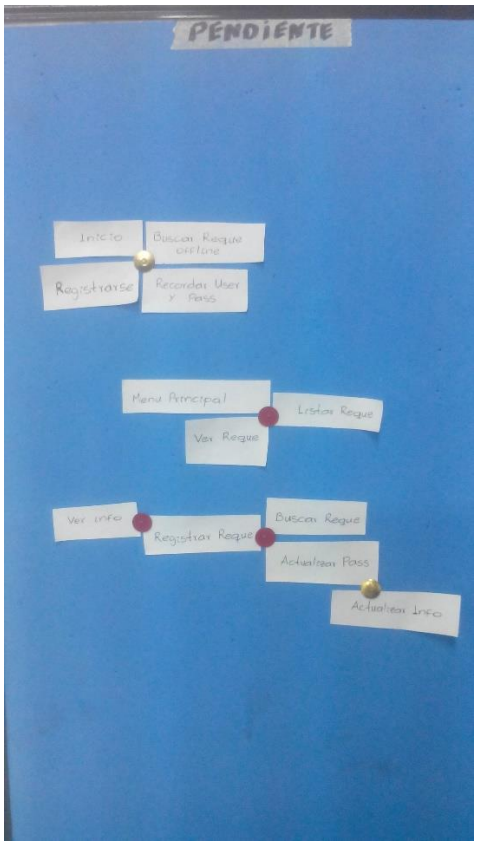


Figura 4.16

StoryImplementación

A continuación se ilustra una parte del flujo de la implementación en el tablero, también unas imágenes de la interfaz gráfica en el estado completado (ver Figura 4.17 – Figura 4.20), estas vistas o interfaces gráficas se han implementado mediante páginas HTML con el apoyo del Framework JQuery Mobile.

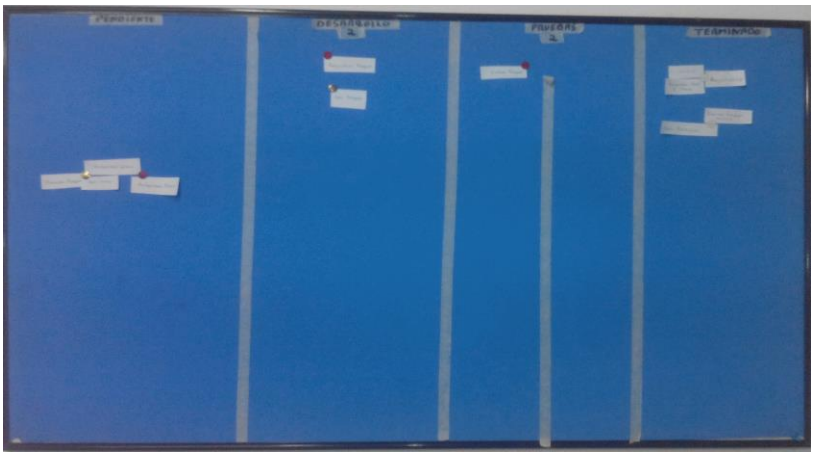


Figura 4.17

Figura 4.18

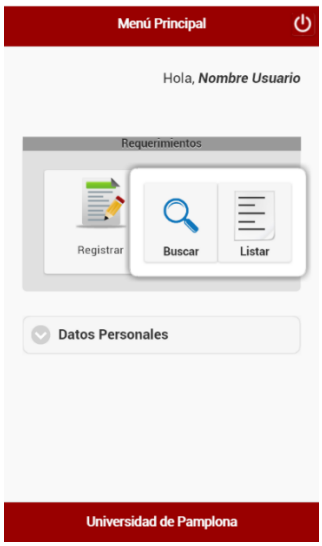


Figura 4.19

Figura 4.20

StoryIntegración

La integración lleva el mismo orden de secuencia que la integración realizada en la Etapa anterior, en este caso los botones denotan la dependencia entre las interfaces gráficas.

PruebaAceptación

En esta ocasión el dueño del producto o cliente aceptó el diseño ya implementado de la aplicación móvil, puesto que ya había aceptado este diseño en la Etapa anterior, aunque en esta ocasión quedó mucho más complacido.

Producción de Servicios:

StoryOrganización

La organización de los servicios a implementar es semejante al proceso realizado en la fase de Producción de Vistas, por dichos motivos se ha obviado la ilustración en esta fase del proceso.

StoryImplementación

La implementación de cada uno de los servicios pertenecientes a las vistas anteriormente implementadas se ha desarrollado en el lenguaje de programación Java, utilizando el IDE NetBeans (versión 8.0.2). Para desarrollar los servicios que están alojados en un servidor donde la aplicación móvil va a realizar sus respectivas solicitudes, se ha utilizado un estilo de arquitectura REST (Representational State Transfer) (ver Anexo).

4.2.4 Etapa AppAjuste

Se han llevado a cabo una serie de ajustes en cuanto a la interfaz gráfica y la simplificación de código.

AspectoElegante

Se han realizado unas leves mejoras en la estructura de la interfaz gráfica (ver Figura 4.21 – Figura 4.24):

Figura 4.21

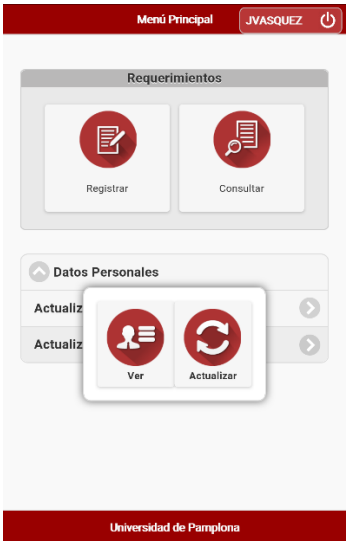


Figura 4.22

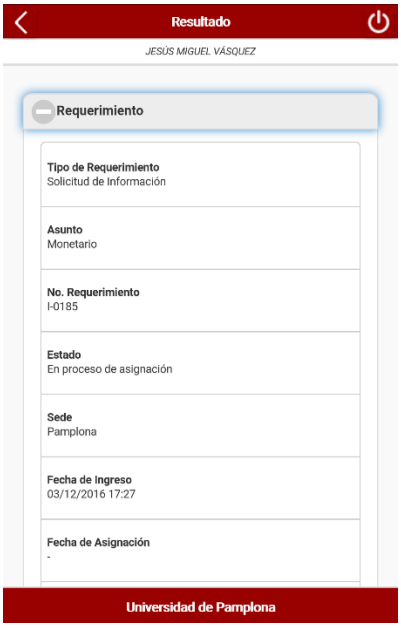


Figura 4.23

Figura 4.24

4.2.5 Etapa AppPrueba

Se han realizado unas series de pruebas finales con el objetivo de comprobar que todas las partes de la aplicación ya integradas funcionan correctamente.

AplicaciónPrueba

El guion de pruebas realizado para examinar la aplicación móvil es el siguiente:

1. Funcionalidades o servicios establecidos en el manual de la aplicación del sistema de PQRDS – Rol Usuario – de la Universidad de Pamplona.
   - Validación inicio de sesión para el usuario.
   - Registrar como nuevo usuario.
   - Recordar nombre de usuario, o recordar usuario y contraseña.

- Buscar requerimiento no en línea (cuando ha registrado una petición, queja, etc. a la Universidad por un medio diferente como puede ser de manera presencial en una oficina, por fax, entre otros).
- Registrar un nuevo requerimiento.
- Listar todos los requerimientos registrados por el usuario.
- Buscar uno o varios requerimientos filtrados por algún componente del mismo (ejemplo por fechas, asuntos, etc.).
- Ver los detalles de un requerimiento (información general, funcionario público asignado, respuesta al requerimiento, etc.).
- Modificar contraseña.
- Ver datos personales.
- Modificar datos personales.

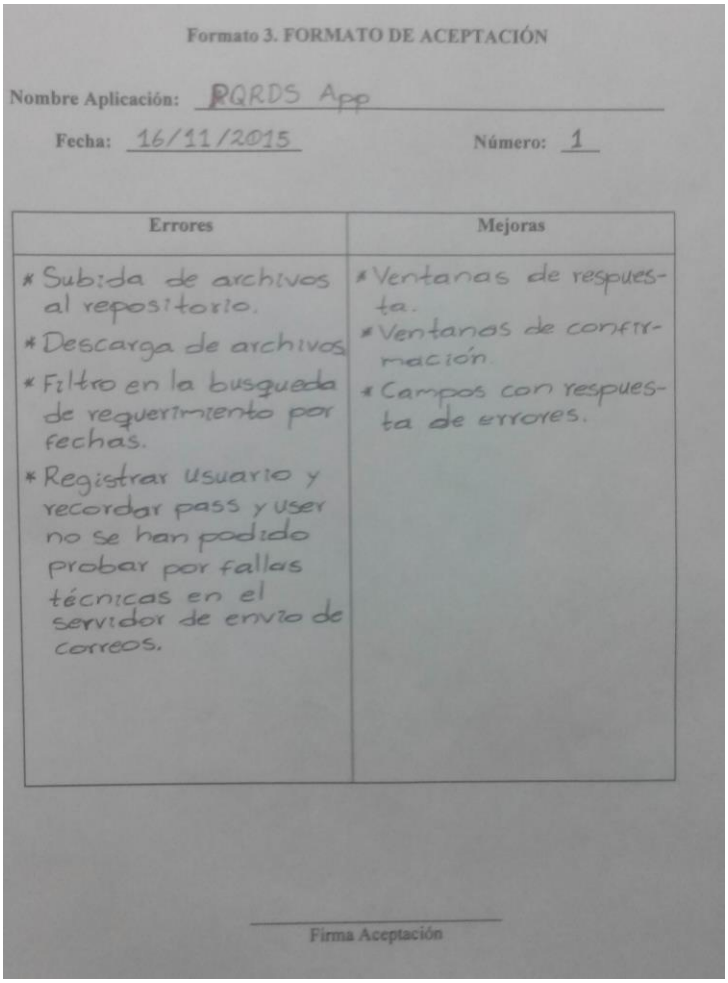
2. Funcionamiento de los servicios de la aplicación móvil durante un periodo de tiempo prolongado (entre 5 y 10 minutos).
3. Usabilidad, probado por personas ajenas al funcionamiento de esta nueva aplicación.
4. Consumo de recursos del teléfono inteligente y valoración de la eficiencia por parte de la aplicación.
5. Instalación y funcionamiento de la aplicación en diferentes teléfonos inteligentes.

AnálisisPrueba

El **primer** y **segundo** punto del guion anterior, han funcionado correctamente. El **tercer** punto tuvo una duda de manera generalizada con respecto a la palabra "requerimiento" en el primer instante de utilización de la app, pero a pesar de ese leve inconveniente se considera aprobado el tercer punto. En el **cuarto** punto, el consumo de recursos está dentro de lo normal (tamaño de aproximadamente 5 mb, consumo de caché de aproximadamente 10 kb, etc.), en cuanto a la eficiencia depende de la velocidad que le ofrece la red de servicio de internet a la que se encuentre conectado el dispositivo móvil debido a que esta aplicación necesita acceso a internet. En el **quinto** punto, la aplicación presentó serios inconvenientes en algunos dispositivos relacionado con el código fuente implementado en el lenguaje interpretado JavaScript, se procedió a retornar a la Etapa AppProducción para realizar los respectivos ajustes.

PruebaAceptación

El dueño de producto o cliente en primera instancia ha solicitado un ajuste en los servicios de carga y descarga de archivos, y en el filtro de búsqueda de requerimientos (ver Figura 4.25), no ha tenido problemas en aceptar el resto de funcionamiento de la aplicación móvil debido a que ya conocía en gran parte la aplicación gracias a las revisiones anteriores que había realizado.



Formato 3. FORMATO DE ACEPTACIÓN

Nombre Aplicación: PQRDS App

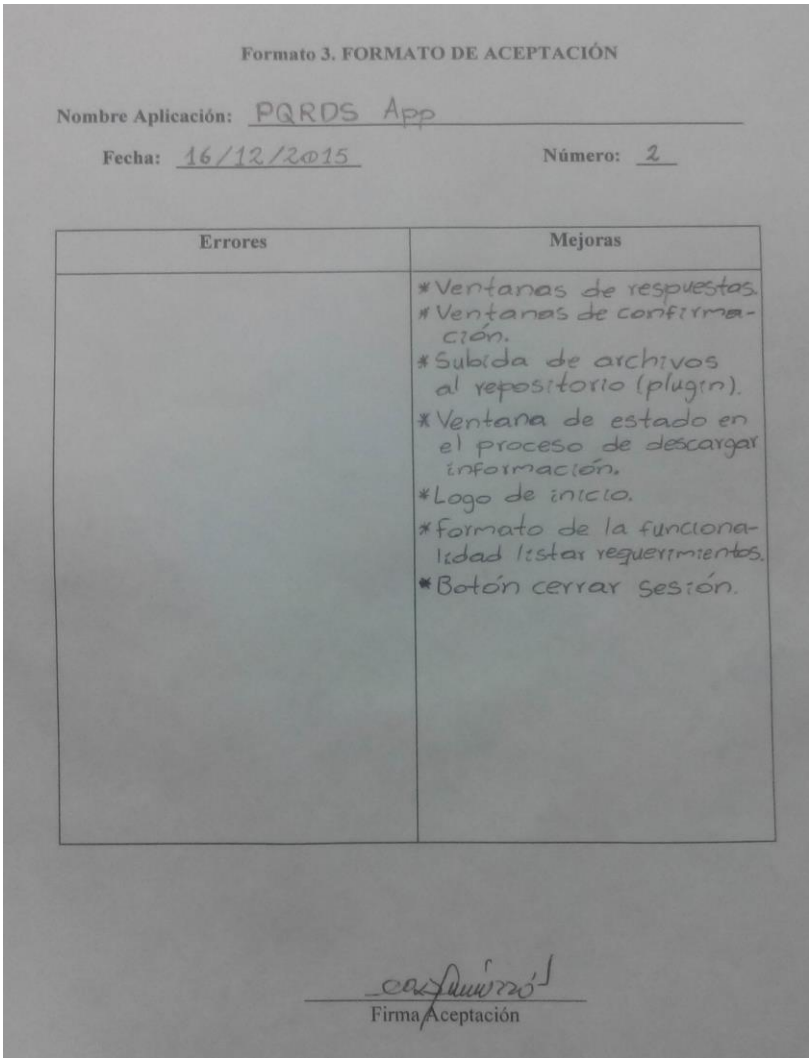
Fecha: 16/11/2015 Número: 1

Errores	Mejoras
* Subida de archivos al repositorio. * Descarga de archivos. * Filtro en la busqueda de requerimiento por fechas. * Registrar usuario y recordar pass y user no se han podido probar por fallas técnicas en el servidor de envio de correos.	* Ventanas de respuesta. * Ventanas de confirmación. * Campos con respuesta de errores.

Firma Aceptación

Figura 4.25

Debido a que el Ingeniero Wilfred Villalba no se encontraba en la ciudad de Pamplona, el coordinador de toda el área de Desarrollo Tecnológico del CIADTI, Ingeniero Elvis Navarro Vega procedió a la firma del formato de aceptación de la aplicación, identificando las mejoras que debe tener esta aplicación (ver Figura 4.26).



Formato 3. FORMATO DE ACEPTACIÓN

Nombre Aplicación: PQRDS App

Fecha: 16/12/2015 Número: 2

Errores	Mejoras
	* Ventanas de respuestas. * Ventanas de confirmación. * Subida de archivos al repositorio (plugin). * Ventana de estado en el proceso de descargar información. * Logo de inicio. * formato de la funcionalidad listar requerimientos. * Botón cerrar sesión.

Firma Aceptación

Figura 4.26

CONCLUSIONES

Se realizó una revisión bibliográfica por las diferentes metodologías ágiles de desarrollo de software, analizando sus características y los aportes que estas agregaron al diseño del procedimiento.

No existe una metodología ágil de desarrollo de software que se ajuste para todos los tipos de proyectos, cada metodología es útil siempre y cuando se adapte al proyecto a desarrollar.

Es cierto que hay variedad de metodologías ágiles de desarrollo de software que son útiles para múltiples proyectos, pero a veces es necesario utilizar conceptos de distintas metodologías y así formar una especie de híbrido para ajustarla a proyectos con características especiales, como es producir una aplicación móvil.

Se diseñó un procedimiento basado en las características de las metodologías identificadas, el procedimiento también se ajustó a la forma de trabajo que está establecida en la oficina de Desarrollo Específico ubicada en el CIADTI.

Se desarrolló la aplicación móvil del sistema de PQRDS de la Universidad de Pamplona acorde a los requerimientos del CIADTI aplicando el procedimiento realizado y cumpliendo lo esperado en el desarrollo de la práctica empresarial.

Por otro lado se ha tenido varias dificultades en la implementación de la interfaz gráfica y de los servicios de la aplicación móvil del sistema de PQRDS de la Universidad de Pamplona, gracias a la necesidad de incorporar nuevos conceptos y tecnologías para realizar dicha implementación; pero con la satisfacción de que esas dificultades han provocado el incremento de conocimiento hacia el desarrollador gracias a esos nuevos conceptos de programación.

La validación del procedimiento se realizó con el desarrollo de la aplicación móvil y la posterior revisión de los ingenieros del CIADTI donde avalaron el resultado y entregaron criterios favorables en la aplicación del procedimiento.

RECOMENDACIONES

El presente procedimiento desarrollado en este proyecto para la implementación de aplicaciones para dispositivos móviles debe implementarse en el CIADTI ya que se ajusta a los procesos que allí se llevan a cabo para el desarrollo o implementación de aplicaciones móviles.

Este procedimiento también debería ser adoptado por los estudiantes de pregrado del programa de Ingeniería de Sistemas de la Universidad de Pamplona, como una guía para apoyar el proceso de aprendizaje en el desarrollo de aplicaciones para dispositivos móviles.

La aplicación móvil del sistema de PQRDS está diseñada como una herramienta para las reclamaciones, quejas, sugerencias, entre otros. De igual forma se puede utilizar esta misma aplicación para recomendar acciones de mejoras en la calidad del servicio prestado por esta.

BIBLIOGRAFÍA

Abud Figueroa, M. A. (2004). *Calidad en la Industria del Software. La Norma ISO-9126.* Obtenido el 17 de septiembre de 2015 de 148.204.210.204/revistaupiicsa/34/34-2.pdf

Amaya Balaguera, Y. D. (2013). Metodologías ágiles en el desarrollo de aplicaciones para dispositivos móviles. Estado actual. *Revista de Tecnología*, 111-124. Obtenido el 2 de junio de 2015 de http://www.uelbosque.edu.co/sites/default/files/publicaciones/revistas/revista_tecnologia/volumen12_numero2/12Articulo_Rev-Tec-Num-2.pdf

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., . . . Thomas, D. (2001). *Manifiesto por el Desarrollo Ágil de Software*. Obtenido el 2 de junio de 2015 de http://www.agilemanifesto.org/iso/es/

Blanco, P., Camarero, J., Fumero, A., Werterski, A., & Rodríguez, P. (2009). *Metodología de desarrollo ágil para sistemas móviles. Introducción al desarrollo con Android y el iPhone.* Madrid. Obtenido el 2 de junio de 2015 de http://www.adamwesterski.com/wp-content/files/docsCursos/Agile_doc_TemasAnv.pdf

Bozheva de Berriprocess, T. (2013). Kanban: 6 Prácticas para Aumentar la Eficiencia en Proyectos TIC. 490-494. Obtenido el 20 de agosto de 2015 de http://www.revistadyna.com/Recursos/Controles/descarga.aspx?IdDocumento=6955&Tipo=1&CodIdioma=&IdWeb=e8d948e0-b75e-4537-8e16-687622b6b7ce

Cortés Morales, R. (s.f). *Introducción al Análisis de Sistemas y la Ingeniería de Software.* Editorial Universidad Estatal a Distancia. Obtenido de https://books.google.com.co/books?isbn=9977649618

Gamboa Manzaba, J. (2014). Aumento de la productividad en la gestión de proyectos, utilizando una metodología ágil aplicada en una fábrica de software en la ciudad

de Guayaquil. *Revista Tecnológica ESPOL*, 1-36. Obtenido el 1 de junio de 2015 de http://www.rte.espol.edu.ec/index.php/tecnologica/article/view/312

Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, P. (s.f). *Metodología de la investigación - Quinta edición.* McGraw-Hill.

Joskowicz, J. (2008). *Reglas y Prácticas en eXtreme Programming.* Obtenido el 6 de junio de 2015 de http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf

MinTic. (2012). *Estrategia de Gobierno en Línea.*

Monge Fallas, J. M. (2012). *Metodologías ágiles aplicadas a la Administración de Proyectos de Desarrollo de Software.* Obtenido el 1 de junio de 2015 de bb9.ulacit.ac.cr/tesinas/publicaciones/044259.pdf

Orjuela Duarte, A., & Rojas, M. (2008). Las Metodologías de Desarrollo Ágil como una Oportunidad para la Ingeniería del Software Educativo. *Revista Avances en Sistema e Informática*, 160-171. Obtenido el 1 de junio de 2015 de http://www.bdigital.unal.edu.co/15430/1/10037-18216-1-PB.pdf

Ramsin, R. (s.f.). Agile Methodologies: Crystal., (págs. 1-14). Obtenido el 18 de septiembre de 2015 de http://sharif.edu/~ramsin/index_files/sdmlecture12.pdf

Schwaber, K., & Sutherland, J. (2013). *La Guía de Scrum.* Obtenido el 22 de agosto de 2015 de http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-ES.pdf

Services, O. o. (Revalidated: 2008). Selecting a development approach. Obtenido el 3 de junio de 2015 de https://www.cms.gov/research-statistics-data-and-systems/cms-information-technology/xlc/downloads/selectingdevelopmentapproach.pdf

Sommerville, I. (2005). *Ingeniería del Software. Septima edición.* Madrid: Pearson Educación. Obtenido de https://books.google.com.co/books?isbn=8478290745

ANEXOS

1. FUNCIONAMIENTO DEL PROCESO CLIENTE-SERVIDOR

El cliente, que en este caso está constituido por páginas “HTML” se comunica con el servidor mediante la tecnología “Ajax” a través de llamadas utilizando el formato “JSON”. El “Servidor” se encarga de dar respuesta a la solicitud realizada por su respectivo cliente (ver Imagen 1).

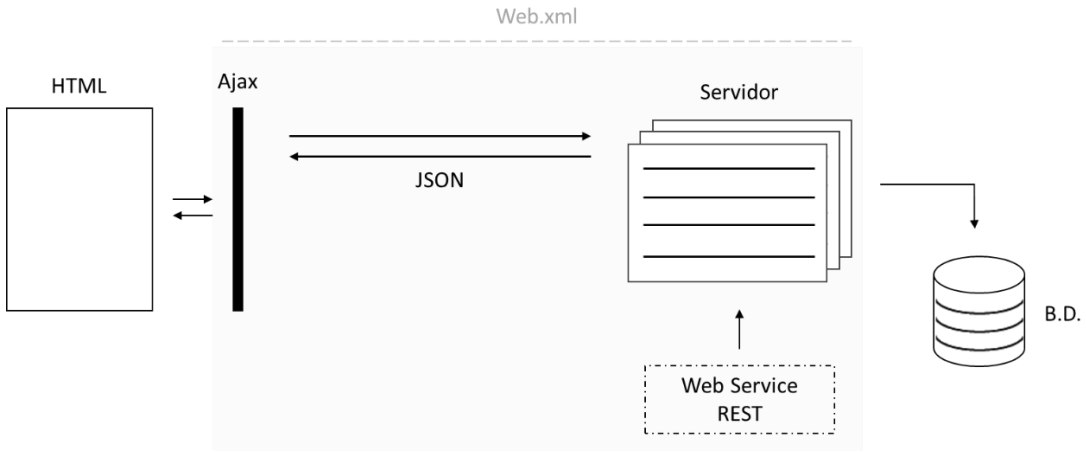


Imagen 1

El “Servidor” está diseñado por medio de servicios web (Web Service), estos servicios están implementados utilizando un estilo de arquitectura REST que se describe a continuación:

Se crea una o varias clases donde se van a utilizar las operaciones de la API REST: GET (para consumir recursos en el servidor – ver Imagen 2) y/o POST (para producir y consumir recursos – ver Imagen 3).

```
6   package ejemplo.serviciosWeb;
7
8   import javax.ws.rs.GET;
9   import javax.ws.rs.Path;
10  import javax.ws.rs.Produces;
11  import javax.ws.rs.core.MediaType;
12  import javax.ws.rs.core.Response;
13
14  /**
15   *
16   * @author JMSIERRA
17   */
18  @Path( "/primer" )
19
20  public class prueba
21  {
22      @GET
23      @Path( "/servicio" )
24      @Produces( MediaType.APPLICATION_JSON )
25
26      public Response primeraImpresion()
27      {
28          return Response.ok( "Hola Mundo" ).build();
29      }
30  }
```

Imagen 2

```
6   package ejemplo.serviciosWeb;
7
8   import ejemplo.ejemploVO.PersonaVO;
9   import javax.ws.rs.Consumes;
10  import javax.ws.rs.POST;
11  import javax.ws.rs.Path;
12  import javax.ws.rs.Produces;
13  import javax.ws.rs.core.MediaType;
14  import javax.ws.rs.core.Response;
15
16  /**
17   *
18   * @author JMSIERRA
19   */
20  @Path( "/registrar" )
21
22  public class RegistroPersona
23  {
24      @POST
25      @Path( "/nuevaPersona" )
26      @Produces( MediaType.APPLICATION_JSON )
27      @Consumes( MediaType.APPLICATION_JSON )
28
29      public Response registrarPersona( PersonaVO perVO )
30      {
31          PersonaVO datoPer = new PersonaVO();
32          datoPer.setPer_id( perVO.getPer_id() );
33          datoPer.setPer_nombres( perVO.getPer_nombres() );
34          datoPer.setPer_profesion( perVO.getPer_profesion() );
35          datoPer.setPer_descripcion( perVO.getPer_descripcion() );
36
37          return Response.ok( datoPer ).build();
38      }
39  }
```

Imagen 3

Estas operaciones en Java las podemos utilizar importando el conjunto de herramientas "javax.ws.rs.*" de la API Jersey, los servicios creados tienen un identificador único global "*http://localhost:8080/AppWeb/servicios/primer/servicio* para el ejemplo de la Imagen 2" que es utilizado por el cliente (en este caso la página HTML) para accederlo.

El cliente localiza el identificador único global del servicio gracias a una configuración realizada en el contexto de la aplicación ubicada en el servidor, específicamente en un archivo llamado "web.xml" ubicado en el directorio "WEB-INF", donde se inicializa el parámetro con el nombre del paquete donde se encuentran las clases que prestan los servicios (ver Imagen 4), también se establece la url padre (todos los identificadores globales tienen la misma estructura hasta esta instancia).

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2
3  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:w
4
5      <servlet>
6          <servlet-name>Jersey REST Service</servlet-name>
7          <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
8          <init-param>
9              <param-name>jersey.config.server.provider.packages</param-name>
10             <param-value>ejemplo.serviciosWeb</param-value>
11         </init-param>
12         <init-param>
13             <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>
14             <param-value>true</param-value>
15         </init-param>
16         <load-on-startup>1</load-on-startup>
17     </servlet>
18
19     <servlet-mapping>
20         <servlet-name>Jersey REST Service</servlet-name>
21         <url-pattern>/servicios/*</url-pattern>
22     </servlet-mapping>
23
24     <session-config>
25         <session-timeout>30</session-timeout>
26     </session-config>
27
28 </web-app>
```

Imagen 4

El cliente realiza las peticiones al servidor mediante la tecnología Ajax, que a su vez realiza el intercambio de datos mediante el formato JSON (ver Imagen 5 – Imagen 6).

```
var servidorURL = "http://localhost:8080/AppWeb/servicios";

$( document ).ready( function()
{
    cargarInformacion();
});

function cargarInformacion()
{
    $.ajax({
        url: servidorURL+ "/datos/persona",
        type: "GET",
        contentType: "application/json",
        timeout: 100000,
        dataType: "json"

    }).done( function( data ){

        var objeto = JSON.parse( JSON.stringify( data ) );

        mostrarInformacion( objeto );

    }).fail( function( jqXHR, textStatus, errorThrown ){

        alert( "No se pueden cargar la información." );

    }).always( function(){

    });
}

function mostrarInformacion( datos )
{
    var textoHtml = '';

    $( "#nombre" ).text( datos.per_nombres );
    $( "#pro" ).val( datos.per_profesion ).show();

    $( "#des" ).html( '<h2>Descripción</h2>' );
    textoHtml += '<p>' +datos.per_descripcion+ '</p>';

    $( "#des" ).append( textoHtml );
}
```

Imagen 5

```
function registrarPersona()
{
    var filtro = '{ "per_id":"' +document.getElementById( "numId" ).value+ '", "per_nombres":"' +document.getElementById( "nombrePer" ).value+ '" }';

    $.ajax({
        url: servidorURL+ "/registrar/nuevaPersona",
        type: "POST",
        contentType: "application/json",
        timeout: 100000,
        dataType: "json",
        data: filtro

    }).done( function( data ){

        var objetoDatos = JSON.parse( JSON.stringify( data ) );

        alert( "Registro exitoso" );
        mostrarRespuesta( objetoDatos );

    }).fail( function( jqXHR, textStatus, errorThrown ){

        alert( "Error en el registro" );

    }).always( function(){

    });
}
```

Imagen 6