Universidad de Camagüey

Facultad de Informática

Propuesta de metodología de desarrollo de software para su utilización en la línea de productos "Aplicaciones J2ME para la Cultura y el Patrimonio"

Tesis en opción al título de

Máster en Informática Aplicada

Autor:

Ing. Ernesto Avila Domenech

Tutor:

MSc. Enislay Ramentol Martínez

Consultante:

Ing. Abel Meneses Abad

2012

Agradecimientos

A mi consultante Abel por no decirme NO nunca...

A mi tutora Enislay por guiarme en este proceso...

A Yailé por ser ejemplo ante todo...

A la UCI, en especial a la Facultad Regional Granma, por darme la oportunidad de realizar esta investigación...

A mis amigos y compañeros de trabajo por hacer esta investigación más sencilla...

A mi familia por apoyarme en todo momento...

A todos los que de una forma u otra tuvieron que ver con este trabajo...

Dedicatoria

A mis padres por el apoyo incondicional...

A mi hermano por su ayuda...

A mi Ernestico por darme tanta felicidad...

A Lia por aguantarme tanto y acompañarme en todo momento...

Declaración de Autoría

Version 0.1, publicada en abril de 2012.

e-mail: eadomenech@grm.uci.cu, eavila08@graduados.uci.cu

Para que así conste firmo la presente a los ___ días del mes de _____ del año 2012.

Ing. Ernesto Avila Domenech

_____

Autor

Resumen

La Universidad de las Ciencias Informáticas (UCI) creada a raíz de la Batalla de Ideas se encuentra inmersa en una constante actividad productiva. Sus profesores y estudiantes, protagonistas de tan importante actividad, cuentan con las condiciones suficientes y necesarias para desarrollar aplicaciones de alta calidad. A diferencia de otras universidades, los estudiantes que arriben al sexto semestre de la carrera pasan del llamado ciclo de formación básico al ciclo de formación profesional, donde se vinculan directamente a proyectos productivos.

Para una mayor producción, organización y control, la Facultad Regional Granma, perteneciente a la UCI, ha optado por desarrollar todos sus proyectos productivos guiados por la metodología SXP, fusión cubana de las bien conocidas metodologías Scrum y Extreme Programming (XP). Esta estrategia si bien fue un paso de avance, no es del todo eficiente para la totalidad de los proyectos.

En el presente trabajo se realiza un estudio de las metodologías de desarrollo de software más populares y se hace una comparación de acuerdo a sus características para luego emitir una propuesta a utilizar en los proyectos que conforman la línea de productos “Aplicaciones J2ME para la Cultura y el Patrimonio”, dedicada al desarrollo de aplicaciones en la plataforma Java 2 Micro Edition con el objetivo de promover, difundir y divulgar el patrimonio y la cultura de la provincia Granma.

Palabras claves: Ingeniería de Software, J2ME, Metodologías Ágiles, Móvil

Abstract

The Informatics Sciences University (UCI) created following the Battle of Ideas is immersed in a constant production. His teachers and students, as important players in activity, have the necessary and sufficient conditions to develop high quality applications. It differs from other universities, students who arrive on the sixth semester of passing the so-called basic cycle to cycle training, which are directly linked to productive projects.

For maximum production, organization and control, the Regional School Granma, belonging to the UCI, has chosen to develop all their productive projects guided by the methodology SXP, Cuban fusion of the well known methodologies Scrum and Extreme Programming (XP). Although this strategy was a step forward, not quite efficient for all projects.

In the present work a study of software development methodologies most popular and a comparison is made according to their characteristics and then issue a proposal to be used in projects that make up the product line "J2ME applications for Culture and heritage", dedicated to developing applications in Java 2 Micro Edition platform with the aim to promote, publicize and disseminate the heritage and culture of the province of Granma.

Keywords: Software Engineering , J2ME, Agile Methodologies , Mobile.

Tabla de Contenidos

Índice

Índice de ilustraciones

Índice de tablas

Introducción

Dentro de la industria cubana del software, la Universidad de las Ciencias Informáticas (UCI), juega un papel principal pues combina docencia y producción con el objetivo de convertirse en una universidad innovadora de excelencia científica, distinguida por el uso de las tecnologías. En la UCI se produce una considerable cantidad de software que contribuyen a la informatización de los procesos de la universidad, así como a muchas esferas del país y de otras naciones latinoamericanas, elevando así los ingresos por conceptos de producción de software.

La UCI tiene como misión formar ingenieros integrales en el área de la informática, los cuales cursan sus asignaturas de pre-grado en curso regular diurno, pero con una vinculación directa en la producción de software, de manera que la formación de los mismos sea enriquecida con la experiencia práctica. Un estudiante UCI es formado a lo largo de cinco años y una vez culminada su carrera con la discusión de un Trabajo de Diploma es ubicado a prestar sus servicios en una institución estatal que los haya solicitado. [1]

En el curso 2006-2007 se crearon tres facultades regionales en tres provincias del país: La Habana (actual Artemisa), Ciego de Ávila y Granma. Dichas facultades surgieron con el objetivo fundamental de incrementar en cada una de las regiones del país el número de graduados en la carrera Ingeniería en Ciencias Informáticas.

En la inauguración, el entonces rector de la universidad Melchor Gil Morell, expresó que estos centros no solo tienen el objetivo de formar expertos de nivel superior en la rama de la Informática, sino también contribuir al desarrollo de esa ciencia en las localidades, mediante la superación de cuadros y especialistas y su participación en programas de desarrollo provinciales y municipales. [2]

En particular la Facultad Regional Granma (FRG) abrió sus puertas el 4 de abril de 2007 en el municipio Manzanillo, contando en un inicio con 185 estudiantes.

En la actualidad, los estudiantes y profesores de la FRG desarrollan más de 20 proyectos de diferentes tipos, los cuales se han asignado a una de las cuatro líneas de productos que a continuación se mencionan:

1. **Sistemas Integrales de Gestión (SIG):** Reúne todos los proyectos de gestión.

Un sistema de gestión es una estructura probada para la gestión y mejora continua de las políticas, los procedimientos y procesos de la organización. También se incluyen, en esta línea de productos, los proyectos donde se hace necesario trabajar con información georeferenciada; en dichos proyectos es inminente el manejo de una gran cantidad de elementos espaciales vinculados de una forma u otra con diversas variables y parámetros.

2. **Realidad Virtual (RV):** Agrupa los proyectos donde existan elementos de realidad virtual, ejemplo de ello son los paseos y galerías virtuales en la web.
3. **Web y Multimedia (WM):** Abarca los proyectos relacionados con portales y multimedias.
4. **Aplicaciones J2ME para la Cultura y el Patrimonio (AppJ2ME):** la componen los proyectos que utilizan la plataforma Java 2 Micro Edition (J2ME) para el desarrollo de aplicaciones para dispositivos móviles y que están enfocados en la promoción y difusión de la cultura y el patrimonio de la provincia Granma.

Es importante mencionar que esta última línea surgió gracias al avance de la tecnología y de su gran aceptación. Los dispositivos móviles, son ejemplo de ello y se han convertidos en objetos inseparables para los seres humano.

Se definirá como dispositivos móviles, a aquellos dispositivos, que por su naturaleza física o virtual, puedan desplazarse de un lugar a otro conservando sus parámetros de configuración y funcionalidad. Con el fin de delimitar el subconjunto de dispositivos a los que se refieren en este informe se dirá, además, que son aquellos diseñados a los efectos de acompañar a sus usuarios en sus actividades de rutina. PDAs (Personal Digital Assistant, en español Asistente Digital Personal), Handhelds y teléfonos celulares, podrían ser ejemplos de estas tecnologías. [3]

En un principio el grupo que desarrollaba aplicaciones en la plataforma J2ME era bien reducido; además, sus integrantes no utilizaban en lo absoluto una metodología de desarrollo de software. Por tal motivo las primeras aplicaciones entregadas, si bien eran llamativas y novedosas, no poseían una calidad aceptable.

Unos pocos meses después de comenzar a desarrollar estas aplicaciones, la dirección

de la Facultad Regional Granma definió, como parte de la gestión de configuración, emplear la metodología SXP (fusión cubana de las bien conocidas metodologías Extreme Programming y Scrum) para el desarrollo de todos los proyectos pertenecientes a cada una de las líneas de productos. Sin duda alguna fue un paso de avance, pues se venía desarrollando de forma desorganizada y por diferentes personas con escasa comunicación.

Con el transcurso del tiempo se evidenció un gran avance en todos los proyectos así como un aumento de la calidad de los mismos, aunque se notó que la metodología empleada no se adaptaba del todo en los proyectos de la línea de productos “Aplicaciones J2ME para la Cultura y el Patrimonio”; pues en su diseño no se tuvo en cuenta la diferencia existente entre el desarrollo de aplicaciones para móviles y el desarrollo del software tradicional.

El desarrollo de aplicaciones móviles difiere del desarrollo de software tradicional en muchos aspectos, lo que provoca que las metodologías usadas para estos entornos también difieran de las del software clásico. Esto es porque el software móvil tiene que satisfacer una serie de requerimientos y condicionantes especiales que lo hace más complejo. [4]

1. **Canal radio.** Consideraciones tales como la disponibilidad, las desconexiones, la variabilidad del ancho de banda, la heterogeneidad de redes o los riesgos de seguridad han de tenerse especialmente en cuenta en este entorno de comunicaciones móviles.
2. **Movilidad.** Aquí influyen consideraciones como la migración de direcciones, alta latencia debido a cambio de estación base o la gestión de la información dependiente de localización. Sobre esta última, de hecho, se pueden implementar un sinfín de aplicaciones, pero la información de contexto asociada resulta muchas veces incompleta y varía frecuentemente.
3. **Portabilidad.** La característica portabilidad de los dispositivos terminales implica una serie de limitaciones físicas directamente relacionadas con el factor de forma de los mismos, como el tamaño de las pantallas (algo que ha variado sustancialmente con la popularización de las pantallas táctiles), o del teclado, limitando también el número de teclas y su disposición.

4. **Fragmentación de la industria.** La existencia de una considerable variedad de estándares, protocolos y tecnologías de red diferentes añaden complejidad al escenario del desarrollo móvil.
5. **Capacidades limitadas de los terminales.** Aquí se incluyen factores como la baja potencia de cálculo o gráfica, los riesgos en la integridad de datos, las interfaces de usuario poco funcionales en muchos aspectos, la baja capacidad de almacenamiento, la duración de las baterías o la dificultad para el uso de periféricos en movilidad. Factores todos que, por otro lado, están evolucionando en la dirección de la convergencia de los ultraportátiles (netbooks) con los dispositivos inteligentes (smartphones) constituyendo cada vez menos un elemento diferencial.
6. **Diseño.** Desde el punto de vista del desarrollo, el diseño multitarea y la interrupción de tareas es clave para el éxito de las aplicaciones de escritorio; pero la oportunidad y frecuencia de éstas es mucho mayor que en el software tradicional, debido al entorno móvil que manejan, complicándose todavía más debido a la limitación de estos dispositivos.
7. **Usabilidad.** Las necesidades específicas de amplios y variados grupos de usuarios, combinados con la diversidad de plataformas tecnológicas y dispositivos, hacen que el diseño para todos se convierta en un requisito que genera una complejidad creciente difícil de acotar.
8. **Time-to-market.** En un sector con un dinamismo propio, dentro de una industria en pleno cambio, los requisitos que se imponen en términos de tiempo de lanzamiento son muy estrictos y añaden no poca dificultad en la gestión de los procesos de desarrollo.

Al no tenerse en cuenta estos aspectos en la metodología empleada, ocurrieron algunos inconvenientes en el desarrollo de los pocos productos implementados en la línea. Los más importantes fueron:

1. El equipo de desarrollo demoró como promedio ocho meses para implementar aplicaciones sencillas. Ejemplo de ello fue la Galería Virtual “Vacas” del pintor Roberto Reytor. **(Ver Anexo A)**

2. En algunas ocasiones los integrantes del equipo de desarrollo se sintieron incómodos pues tenían que entregar artefactos, según la metodología empleada, que consideraban innecesarios.
3. Al no existir una etapa bien definida de comercialización no existía una buena retroalimentación con los usuarios finales.
4. Al no existir un cliente físico quedaba empañada la práctica “Cliente in-situ”, práctica que propone mantener al cliente en el lugar donde se desarrolla la aplicación.
5. Los Sprints de 30 días eran muy largos y atrasaba la salida de los productos Betas al mercado.

Por lo antes expuesto se decidió realizar un profundo estudio de las metodologías de desarrollo de software más populares y utilizadas, para luego desarrollar una nueva con el fin de guiar específicamente el proceso de desarrollo de software en la línea de productos “Aplicaciones J2ME para la Cultura y el Patrimonio” perteneciente a la Facultad Regional Granma. Por lo tanto el **problema** de esta investigación queda definido como: La metodología de desarrollo de software que se utiliza en la línea de productos “Aplicaciones J2ME para la Cultura y el Patrimonio” no se adecúa a las características de sus proyectos por lo que hace engorroso el trabajo de los mismos afectando el proceso de desarrollo.

El **objeto de estudio** de esta investigación lo constituye el proceso de desarrollo de software en líneas de productos para dispositivos móviles. Específicamente el **campo de acción** está relacionado con la metodologías de desarrollo de software utilizadas en las mismas. El **objetivo genera**l de la investigación es proponer una metodología de desarrollo de software que se adecúe a las características de los proyectos que conforman la línea de productos “Aplicaciones J2ME para la Cultura y el Patrimonio” perteneciente a la Facultad Regional Granma y que evaluándola mediante métodos existente en la literatura se obtengan resultados satisfactorios. Dicho objetivo general está desglosado en los siguientes **objetivos específicos**:

1. Realizar la fundamentación teórica para justificar la investigación y dejar definida la posición del autor.

2. Proponer una metodología para guiar el proceso de desarrollo de software en los proyectos pertenecientes a la línea de productos “Aplicaciones J2ME para la Cultura y el Patrimonio”.
3. Evaluar la metodología propuesta mediante métodos evaluadores de metodologías de desarrollo de software existentes en la literatura.

Métodos de investigación

Durante el desarrollo de la investigación fueron utilizados métodos teóricos que a continuación se mencionan:

Lógicos

Hipotético–Deductivo: para la elaboración de la idea central de la investigación y para proponer nuevas líneas de trabajo a partir de los resultados parciales; este método permite llegar a conclusiones a partir de los conocimientos acumulados.

Sistémico: para el estudio del objeto a través de sus componentes y las relaciones entre ellos.

Dialéctico: para la determinación de las contradicciones existentes que caracterizan el comportamiento del objeto y así llegar a la solución del problema planteado. Este método tiene la ventaja de interrelacionar tanto el objeto como el sujeto.

Métodos investigativos particulares

La entrevista: con el objetivo de obtener información sobre los grupos de desarrollo, específicamente relacionado con el proceso de producción en el que se encuentran vinculados.

La encuesta: para obtener información sin la intervención del investigador, con un cuestionario más rígido dirigido a toda la muestra.

Estructura de la Tesis

El presente trabajo consta de tres capítulos, conclusiones, recomendaciones, bibliografía, y anexos. Los contenidos de los capítulos, en forma abreviada, son los siguientes:

En el capítulo uno se describe la línea de productos “Aplicaciones J2ME para la Cultura y el Patrimonio” y se realiza una descripción de las principales y más utilizadas metodologías de desarrollo de software, analizando las buenas prácticas que proponen cada una de ellas.

En el segundo capítulo se describe la metodología de desarrollo de software propuesta para guiar los proyectos pertenecientes a la línea de productos “Aplicaciones J2ME para la Cultura y el Patrimonio”.

Por último, en el capítulo tres, se exponen los resultados obtenidos al evaluar la metodología de desarrollo de software propuesta, por dos métodos existentes en la literatura: 4-DAT y un Método Cuantitativo de Evaluación de Metodologías Ágiles. El primero de ellos con características cuantitativas y cualitativas y el segundo puramente cuantitativo.

Capítulo I: Fundamentación Teórica

Introducción

En el presente capítulo se realiza un análisis de las características de la línea de productos “Aplicaciones J2ME para la Cultura y el Patrimonio” perteneciente a la Facultad Regional Granma en la Universidad de las Ciencias Informáticas.

También se aborda sobre el estudio realizado de algunas de las metodologías de desarrollo de software más utilizadas, recalcando las buenas prácticas que proponen.

1.1 Línea de productos “Aplicaciones J2ME para la Cultura y el Patrimonio”

A finales del año 2010 unas pocas personas perteneciente a la Facultad Regional Granma, se dieron a la tarea de crear un juego educativo capaz de ejecutarse en dispositivos móviles. Dicho juego, llamado Puzzle, fue desarrollado en la plataforma Java 2 Micro Edition (J2ME).

El juego consiste en un rompecabezas, en el cual se deben realizar determinados movimientos hasta lograr armar la imagen objetivo. Las imágenes utilizadas visualizan al patriota Carlos Manuel de Céspedes, Padre de la Patria y protagonista del alzamiento en La Demajagua el 10 de octubre de 1868, La Glorieta de Manzanillo (Ilustración 1) y el Ingenio “La Demajagua” (Ilustración 2).

Siendo presentado en diferentes eventos donde la facultad participó, fue adquiriendo popularidad provocando que un considerable número de usuarios solicitaran que fuera enviado a sus móviles para llevarlo a casa y poder mostrárselo a hijos, hermanos y familia en general. Se evidenció, en estos usuarios, una identificación con los sitios que reflejaba la aplicación. Algo que fue altamente considerado por el equipo de desarrollo.

Otro aspecto relevante fue el uso de las nuevas tecnologías de la información y las comunicaciones. En el país (Cuba) se había liberado la venta de líneas para teléfonos celulares en 2008; por lo que la adquisición de dichos móviles de una importante parte de la población era notable.

Considerando este hecho, se aprobó crear un grupo que de manera oficial se especializara en el desarrollo de aplicaciones para estos dispositivos que cada vez son

más utilizados. Ahora todo quedaba en elegir la plataforma de desarrollo a emplear.

Ilustración 1: Puzzle con la imagen La Glorieta

Ilustración 2: Puzzle con la imagen La Demajagua.

Se comenzó a realizar un estudio de las distintas y más importantes plataformas de desarrollo así como los sistemas operativos para dispositivos móviles. Entre los más importantes se encuentran Android, iOS y J2ME.

La propuesta de Google

Android es una solución completa de software de código libre para teléfonos y dispositivos móviles. Es un paquete que engloba un sistema operativo, un "runtime" de ejecución basado en Java, un conjunto de librerías de bajo y medio nivel y un conjunto inicial de aplicaciones destinadas al usuario final (todas ellas desarrolladas en Java). Android se distribuye bajo una licencia libre permisiva (Apache) que permite la integración con soluciones de código propietario. Surge como resultado de la Open Handset Alliance un consorcio de 48 empresas distribuidas por todo el mundo con intereses diversos en la telefonía móvil y un compromiso de comercializar dispositivos móviles con este sistema operativo. El desarrollo viene avalado principalmente por Google (tras la compra de Android Inc. en 2005) y entre las compañías encontramos compañías de software (Ebay, LivingImage...), operadores (Telefónica, Vodafone, T-Mobile...), fabricantes de móviles (Motorola, Samsumg, acer, LG, HTC...) o fabricantes de Hardware (nVidia, Intel o Texas Instruments). [4]

Android rompe las barreras para crear aplicaciones nuevas e innovadoras. Por ejemplo, un desarrollador puede combinar la información de la web con los datos en el

teléfono móvil de un individuo -como los contactos del usuario, calendario o ubicación geográfica – para proporcionar una experiencia de usuario más relevante. Con Android, los desarrolladores pueden crear una aplicación que permite a los usuarios ver la ubicación de sus amigos y ser alertado cuando se encuentran en los alrededores dándoles la oportunidad de conectarse. [5]

Un dispositivo móvil con capacidad para poderle instalar Android, excede de los 100 euros; por lo que sin duda alguna para los residentes en la provincia Granma, provincia oriental de Cuba, y dada la economía del país, resulta demasiado caro. Esto, unido a la inexistencia de un mercado actualizado en este tipo de tecnología, trae como consigo que existan muy pocas personas con dispositivos móviles con Android.

La manzana de Steve Jobs

Si se describe a iOS (anteriormente denominado iPhone OS) se notará un inconveniente similar al de Android. Al igual que Android, iOS es un sistema operativo móvil. Pertenece a Apple y originalmente fue desarrollado para el iPhone (familia de teléfonos inteligentes multimedia con conexión a Internet, pantalla táctil capacitiva y escasos botones físicos), siendo después usado en dispositivos como el iPod Touch, iPad y el Apple TV.

La interfaz de usuario de iOS está basada en el concepto de manipulación directa, usando gestos multitáctiles. Los elementos de control consisten en deslizadores, interruptores y botones. La respuesta a las órdenes del usuario es inmediata y provee una interfaz fluida. La interacción con el sistema operativo incluye gestos como deslices, toques, pellizcos, los cuales tienen definiciones diferentes dependiendo del contexto de la interfaz. Se utilizan acelerómetros internos para hacer que algunas aplicaciones respondan al sacudir el dispositivo (por ejemplo, para el comando deshacer) o rotar en tres dimensiones (un resultado común es cambiar de modo vertical al horizontal).

Si se realiza una comparación entre ambos, obtendremos que en mayo de 2010 en los Estados Unidos, iOS tenía el 59% de consumo de datos móviles (incluyendo el iPod Touch y el iPad) por un 20% Android. Aunque Google OS (Android) creció rápidamente en los meses anteriores, pasando de alrededor del 5% en enero de 2009 al 20% en mayo de 2010, no así iOS que cayó del 75% al 59% durante ese mismo período de

tiempo. [6]

En cuanto al precio de los dispositivos donde se pueda instalar iOS se puede decir que es algo similar a los que aceptan Android y quizás hasta más caros llegando la media a los 200 euros. Un ejemplo de ello es el recién lanzado iPad 3 donde las versiones con Wi-Fi de 16, 32 y 64GB de capacidad poseen un precio de 499, 599 y 699 dólares respectivamente y la versión 4G por $629, $729 y $829.

Otra desventaja que tiene iOS sobre Android es su demora en la liberación del código empleado. Hasta marzo de 2008 no se publicó un SDK propiamente dicho para el desarrollo de aplicaciones para el iPhone.

Una plataforma para todos

Otra plataforma de desarrollo es Java 2 Micro Edition (J2ME), la misma permite ampliar su uso a diferentes plataformas de hardware y software en un entorno restrictivo con la instalación de los siguientes componentes: Configuración para Dispositivos con Limitada Conectividad (CLDC) que incluye la maquina virtual (KVM) y el perfil para dispositivos con Información Multimedia (MIDP); MIDP permite, entre otras ventajas, conectividad inalámbrica con diferentes tecnologías como Wi-fi, Bluetooth e IRDA principalmente, manejo de un entorno gráfico de alto nivel, persistencia (RMS) y administración del ciclo de vida de la aplicación. [7]

J2ME es la adición más nueva y pequeña de la familia Java. Es el hermano menor de J2SE (Standard Edition) y la especificación J2EE basada en servidor (Enterprise Edition). J2ME proporciona un entorno de desarrollo para una amplia gama de dispositivos pequeños y limitados. A pesar de que J2ME está dirigida a dispositivos con capacidad limitada, muestra todas las características del lenguaje Java. [8]

Cada combinación de configuración y perfil coincide con un grupo de productos específicamente optimizados para que coincida con la memoria, la potencia de procesamiento y de las capacidades de E / S de cada dispositivo. En la Ilustración 3 se muestra cómo la tecnología se ha desarrollado para ofrecer una plataforma para una serie de circunstancias. Las aplicaciones empresariales se pueden desarrollar utilizando los paquetes de J2EE, aprovechando al máximo la potencia de los servidores de gran tamaño, capaces de transmitir grandes cantidades de datos a través de redes. La edición J2SE complementa J2EE y proporciona la base para

aplicaciones de tipo escritorio. Se puede ver que estas dos versiones de Java se definen con la consideración de la potencia del procesador, la memoria y la capacidad de comunicación. Una inspección más a fondo de la arquitectura Java revela que hay dos grupos de especial interés, bajo la bandera J2ME. J2ME proporciona un entorno para los desarrolladores que deseen desarrollar aplicaciones para dispositivos más pequeños. Este entorno se ha especializado para atender a las máquinas con una capacidad aún menor. [8]

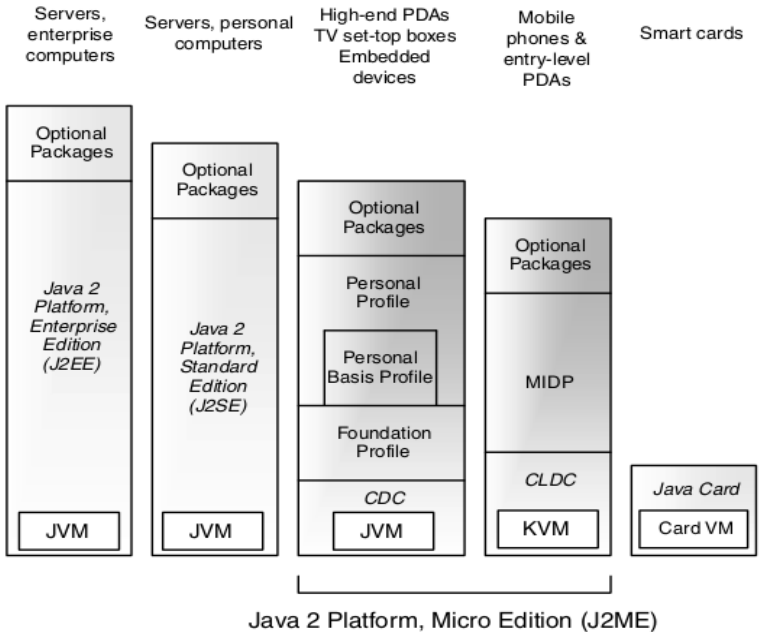

Ilustración 3: Arquitectura de Java (De Jode, 2004)

Una importante decisión

Luego del análisis realizado sobre las distintas plataformas de desarrollo de aplicaciones para dispositivos móviles se decidió utilizar la plataforma J2ME para el desarrollo de las aplicaciones en la línea de productos abordada, debido a que la mayoría de las empresas que elaboran dispositivos móviles lo utilizan como estándar de facto, pretendiendo así, ampliar el campo de acción para este tipo de aplicaciones.

Los productos desarrollados en esta línea, aunque pueden ejecutarse en dispositivos móviles de media y alta gama, están enfocados para los de baja gama; los cuales poseen las siguientes características:

1. Normalmente tienen recursos limitados (poca memoria, potencia de CPU ajustada, etc.) por razones de costo, ya que muchos de ellos se fabrican en grandes cantidades y los ahorros son importantes.
2. A diferencia de los ordenadores de propósito general, poseen un sistema embebido que realiza una serie de tareas determinadas para cumplir unos requerimientos muy específicos.
3. Son parte de la vida diaria y pueden ser de diferentes tipos y tamaños.
4. En periodos relativamente largos no hay ninguna interacción con el usuario, por lo que deben funcionar sin errores y ser capaces de recuperarse por sí mismos en caso de que estos ocurran.

1.2 Metodologías de desarrollo de software

Se define como metodología de desarrollo al marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información. [9]

Se puede decir, para no ser absolutos, que la mayoría de las metodologías ayudan a fortalecer el trabajo en equipo manteniéndolos enfocados en una misma dirección. Además mediante las mismas se puede seguir de forma clara el avance de las tareas a realizar por cada integrante del equipo.

Cuando se compara una con otra, no es para determinar o elegir la superior. No existe una metodología mejor que otra; unas se adaptan más que otras dependiendo del tipo de producto que se desarrolle y según sean las técnicas a utilizar. Además depende de las consideraciones de la organización, proyecto y equipo.

1.2.1 Metodologías tradicionales

Conceptos importantes de las tradicionales. [10]

1. **Mejora de procesos:** un programa de actividades diseñadas para mejorar el rendimiento y madurez de los procesos de la organización, y los resultados de dicho programa. La mejora de procesos surgió de la labor de gestión de calidad de Deming, Crosby y Juran y tiene por objeto aumentar la capacidad de los procesos de trabajo.

2. **Capacidad de proceso:** la capacidad inherente de un proceso para producir los resultados previstos. A medida que la capacidad de cada proceso es mejorada, se hace predecible y mensurable, y las causas más significativas de mala calidad y productividad son controladas o eliminadas. Esto reduce el rango de resultados esperados y reales que se pueden conseguir siguiendo un proceso.

3. **Madurez de organización.** Una organización se dice que está madurando cuando mejora constantemente su capacidad de proceso. Madurez abarca no sólo la capacidad de cada proyecto, también incluye la aplicación común de los procesos estándares a través de la organización.

4. **Grupo en los procesos:** colección de especialistas que faciliten la definición, mantenimiento y mejora del proceso utilizado por una organización. Los grupos de procesos pueden tratar los procesos de ingeniería de software solamente o procesos de ingeniería en general.

5. **Gestión de riesgos:** proceso organizado y analítico para identificar las incertidumbres que pudieran causar daños o pérdida (la identificación de riesgos), evaluar y cuantificar los riesgos identificados, y desarrollar y aplicar planes de gestión de riesgos para prevenir o controlar las causas de riesgo que podrían causar daños o pérdidas significativas.

6. **Verificación.** La verificación confirma que los productos del trabajo (por ejemplo, las especificaciones, diseños, modelos) reflejan adecuadamente los requisitos establecidos para ellos.

7. **Validación.** La validación confirma la idoneidad o el valor de un producto de trabajo de su misión operacional (construyendo el producto correcto).

8. **Arquitectura de sistema de software.** Una arquitectura de sistema de software define una colección de componentes de software y de sistema, los conectores y las limitaciones, una colección de declaraciones de las partes interesadas y una lógica que demuestra que los componentes, conectores, y las limitaciones definidas, en caso de ser aplicadas, satisfacen las necesidades recogidas en las declaraciones de los diferentes actores del sistema.

1.2.1.1 RUP

Kruchten define RUP como un proceso de ingeniería de software, cuyo objetivo es guiar a las organizaciones de desarrollo de software en sus esfuerzos. Se trata de un producto del proceso, diseñado al igual que cualquier producto de software, y se integra con las suites de herramientas Rational de desarrollo de software. RUP tiene una estructura muy bien definida y regular, con un enfoque orientado a objetos para su descripción. [11]

El proceso en sí ha sido diseñado utilizando técnicas similares a las de diseño de software. En particular, tiene un subyacente modelo orientado a objetos, con UML. El proceso consta de dos estructuras o dimensiones:

1. La dimensión horizontal representa el tiempo y muestra los aspectos del ciclo de vida del proceso que se desarrolla.
2. La dimensión vertical representa las disciplinas básicas del proceso (o flujos de trabajo), lo que lógicamente representa actividades del grupo de ingeniería de software, por su naturaleza.

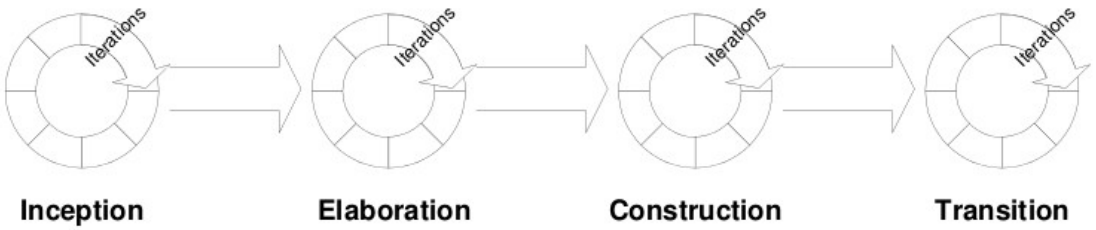

Ilustración 4: Fases de RUP (Abrahamsson, 2002)

La primera (horizontal) representa el aspecto dinámico del proceso que se expresa en términos de ciclos, fases, iteraciones e hitos. En RUP, un producto de software está diseñado y construido en una sucesión de iteraciones adicionales. Esto permite que las pruebas y la validación de ideas de diseño, así como de mitigación de riesgos, ocurran más temprano en el ciclo de vida.

La segunda (vertical) representa el aspecto estático del proceso descrito en términos de componentes de proceso: actividades, disciplinas, artefactos, y roles. [11]

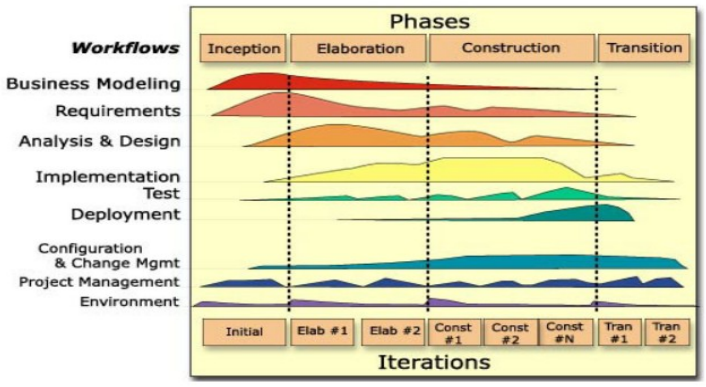

Ilustración 5: Las dos dimensiones de RUP (Kruchten, 2001)

Rational Unified Process captura muchas de las mejores prácticas de desarrollo de software moderno en una forma adecuada para una amplia gama de proyectos y organizaciones:

- Desarrollar software iterativamente.
- Gestión de requisitos.
- El uso de componentes basados en arquitecturas.
- Visualización del modelo de software.
- Continua comprobación de la calidad del software.
- Control de cambios del software.

1.2.2 Metodologías ágiles

En la década de los años 90, surgió un nuevo enfoque que demostraba que no solo con procesos altamente definidos se lograría obtener software en tiempo, costo aceptable y alta calidad. El enfoque fue planteado por primera vez por Martin y se dio a conocer en la comunidad de Ingeniería de Software con el nombre de RAD o Rapid Application Development.

RAD consistía en un entorno de desarrollo altamente productivo, en el que participaban grupos pequeños de programadores utilizando herramientas que generaban código en forma automática tomando como entradas sintaxis de alto nivel. En general, se considera que este fue uno de los primeros hitos en pos de la agilidad en los procesos de desarrollo de software.

Las metodologías ágiles surgen con la creación por parte de Kent Beck, tomando ideas recopiladas junto a Ward Cunningham, de XP (eXtreme Programming). En 1996, Beck es llamado por Chrysler como asesor del proyecto Chrysler Comprehensive Compensation (C3) payroll system. Dada la poca calidad del sistema que se estaba desarrollando, Beck decide tirar todo el código y empezar de cero utilizando las prácticas que él había ido definiendo a lo largo del tiempo. El sistema, que administra la liquidación de aproximadamente 10.000 empleados, y contiene unas 2.000 clases y 30.000 métodos, es puesto en operación en mayo de 1997 casi respetando el calendario propuesto. Como consecuencia del éxito de dicho proyecto, Kent Beck dio origen a XP iniciando el movimiento de metodologías ágiles al que se anexarían otras metodologías surgidas mucho antes que el propio Beck fuera convocado por Chrysler. [12]

Es así como las metodologías de este tipo fueron inicialmente llamadas “metodologías livianas”, sin embargo, aun no contaban con una aprobación pues se le consideraba por muchos desarrolladores como meramente intuitiva. Luego, con el pasar de los años, en febrero de 2001, tras una reunión celebrada en Utah-EEUU, nace formalmente el término “ágil” aplicado al desarrollo de software. En esta misma reunión participan un grupo de 17 expertos de la industria del software, incluyendo algunos de los creadores o impulsores de metodologías de software con el objetivo de esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto. Se pretendía ofrecer una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas.[12]

Tras esta reunión se creó The Agile Alliance, una organización sin ánimo de lucro, dedicada a promover los conceptos relacionados con el desarrollo ágil de software y ayudar a las organizaciones para que adopten dichos conceptos. El punto de partida fue el Manifiesto Ágil, un documento que resume la filosofía “ágil”.

En el Manifiesto Ágil [13] se valora:

1. Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas.

2. Desarrollar software que funciona más que conseguir una buena documentación.
3. La colaboración con el cliente más que la negociación de un contrato.
4. Responder a los cambios más que seguir estrictamente un plan.

Los valores anteriores inspiran los doce principios del manifiesto. Son características que diferencian un proceso ágil de uno tradicional.

Los principios son:

1. La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.
2. Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.
3. Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.
4. La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.
5. Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.
6. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.
7. El software que funciona es la medida principal de progreso.
8. Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.
9. La atención continua a la calidad técnica y al buen diseño mejora la agilidad.
10. La simplicidad es esencial.
11. Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.
12. En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más

efectivo, y según esto ajusta su comportamiento.

Según Boehm y Turner [10] los métodos ágiles poseen los siguientes atributos:

1. **Iterativo:** Varios ciclos.
2. **Incremental:** No se entrega el producto cuando se finalice. Se entregan pequeñas versiones que aunque son capaces de ejecutarse satisfactoriamente no cumplen con todas las funcionalidades exigidas.
3. **Organización propia:** El equipo de desarrollo determina la mejor forma de trabajar.
4. **Emergencia:** Se califican las funcionalidades según su importancia y se desarrollan según este atributo.

1.2.2.1 Extreme Programming (XP)

El énfasis que pone XP en las personas se manifiesta en diversas prácticas que indican que se deben dar más responsabilidad a los programadores para que estimen su trabajo, puedan entender el diseño de todo el código producido y mantengan una metáfora mediante la cual se nombran las clases y métodos de forma consistente. La práctica denominada “semana de 40 horas” indica la necesidad de mantener un horario fijo, sin horas extras ya que esto conlleva al desgaste del equipo y a la posible deserción de sus miembros. Beck afirma que como máximo se podría llegar a trabajar durante una semana con horas extras, pero si pasando ese tiempo las cosas no han mejorado entonces se deberá hacer un análisis de las estimaciones de cada iteración para que estén acordes a la capacidad de desarrollo del equipo.

Si bien XP es la metodología ágil de más renombre en la actualidad, se diferencia de las demás metodologías que forman este grupo en un aspecto en particular: el alto nivel de disciplina de las personas que participan en el proyecto.

El ciclo de vida de XP posee seis fases: Exploración, Planificación, Iteraciones, Producción, Mantenimiento y Muerte. [14]

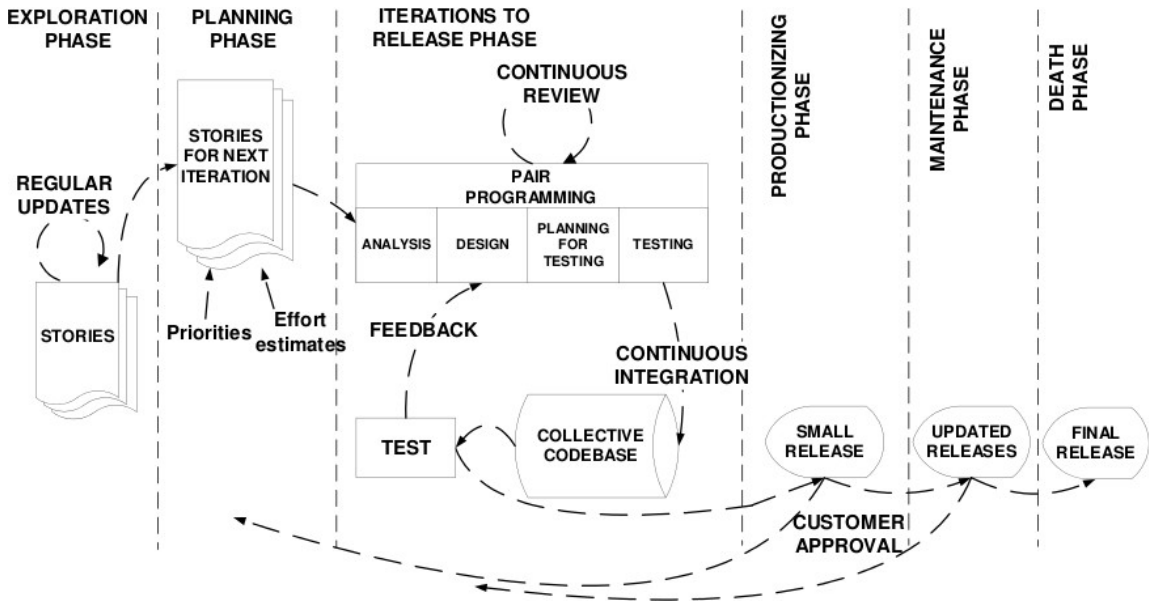

Ilustración 6: Ciclo de vida de XP (Abrahamsson, 2002)

Se pueden resumir las prácticas de XP como sigue: [12]

1. **El juego de la planeación.** Hay una comunicación frecuente entre el cliente y los programadores. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las Historias de Usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración.
2. **Entregas pequeñas.** Producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema. Esta versión ya constituye un resultado de valor para el negocio. Una entrega no debería tardar más de tres meses.
3. **Metáfora.** El sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia compartida que describe cómo debería funcionar el sistema (conjunto de nombres que actúen como vocabulario para hablar sobre el dominio del problema, ayudando a la nomenclatura de clases y métodos del sistema).
4. **Diseño simple.** Se debe diseñar la solución más simple que pueda funcionar y

ser implementada en un momento determinado del proyecto.

5. **Pruebas.** La producción de código está dirigida por las pruebas unitarias. Estas son establecidas por el cliente antes de escribirse el código y son ejecutadas constantemente ante cada modificación del sistema.
6. **Refactorización (Refactoring).** Es una actividad constante de reestructuración del código con el objetivo de remover código duplicado, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar posteriores cambios. Se mejora la estructura interna del código sin alterar su comportamiento externo.
7. **Programación en parejas.** Toda la producción de código debe realizarse con trabajo en parejas de programadores. Esto conlleva ventajas implícitas (menor tasa de errores, mejor diseño, mayor satisfacción de los programadores, etc).
8. **Propiedad colectiva del código.** Cualquier programador puede cambiar cualquier parte del código en cualquier momento.
9. **Integración continua.** Cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día.
10. **40 horas por semana.** Se debe trabajar un máximo de 40 horas por semana. No se trabajan horas extras en dos semanas seguidas. Si esto ocurre, probablemente está ocurriendo un problema que debe corregirse. El trabajo extra desmotiva al equipo.
11. **Cliente in-situ.** El cliente tiene que estar presente y disponible todo el tiempo para el equipo. Este es uno de los principales factores de éxito de XP. El cliente conduce constantemente el trabajo hacia lo que aportará mayor valor de negocio y los programadores pueden resolver de manera inmediata cualquier duda asociada. La comunicación oral es más efectiva que la escrita.
12. **Estándares de programación.** XP enfatiza que la comunicación de los programadores es a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación para mantener el código legible.

1.2.2.2 Scrum

Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle. Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos diez 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 “Sprints”, 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 proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración.

En el ciclo de vida de Scrum cada funcionalidad es escrita en una carta, cada carta es priorizada por el cliente y estimada por el equipo de desarrollo. Luego cada carta es asignada a una iteración. Cuando se implemente, se muestra al cliente y se da como aceptada en caso de satisfacer al mismo.

Prácticas de Scrum:

1. **Product Backlog:** Es donde se define todo lo que se necesita en el producto final, basado en el conocimiento actual. Se compone de una lista de prioridades que constantemente son actualizadas según las necesidades operativas y técnicas para que el sistema esté siendo construido o mejorado.

   Características, funciones, correcciones de errores, defectos, mejoras solicitadas y mejoras tecnológicas son algunos elementos que se pueden incluir en el Product Backlog. Son múltiples los actores que pueden participar en la generación de los elementos del producto, tales como reserva de pedidos de cliente, el equipo de proyecto, marketing y ventas, gestión y atención al cliente. Esta práctica incluye las tareas para crear la lista de reserva del producto, y controlar de forma consistente durante el proceso al agregar, eliminar, especificar, actualizar y dar prioridad a los elementos del Product Backlog. El dueño del producto es el responsable de mantener la lista de reserva del producto.

2. **Effort estimation:** es un proceso iterativo, en el que las estimaciones de los elementos de la reserva del producto se centran en un nivel más preciso debido

a una mayor cantidad de información disponible sobre un tema determinado. El dueño del producto, junto con el Equipo Scrum son los responsables de realizar la estimación del esfuerzo.

3. **Sprint:** Sprint es el procedimiento de adaptación a las necesidades cambiantes de las variables ambientales, tiempo, recursos, conocimientos, tecnología, etc.) El Equipo Scrum se organiza para producir nuevos ejecutables mediante Sprints de aproximadamente treinta días naturales de duración. Las herramientas de trabajo del equipo son las reuniones de planificación de Sprint, la Pila de Sprint y reuniones diarias de Scrum.

4. **Sprint Planning meeting:** es una reunión de dos fases, organizada por el Scrum Master. Los clientes, usuarios, administradores, dueño del producto y el Equipo Scrum participar en la primera fase de la reunión para decidir sobre los objetivos y las funcionalidades del siguiente Sprint. La segunda fase de la reunión se lleva a cabo por el Scrum Master y el Equipo Scrum se centra en cómo el incremento del producto se lleva a cabo durante el Sprint.

5. **Sprint Backlog:** es el punto de partida de cada Sprint. Se trata de una lista de elementos seleccionados del Product Backlog para ser implementados en el próximo Sprint. Los elementos son seleccionados por el Equipo Scrum, junto con el Scrum Master y el dueño del producto en la reunión de planificación de Sprint, sobre la base de los elementos prioritarios y las metas establecidas para el Sprint. A diferencia del Product Backlog, el Sprint Backlog es estable durante el Sprint (es decir, 30 días). Cuando todos los elementos del Sprint Backlog han terminado, una nueva iteración del sistema se entrega.

6. **Daily Scrum meeting:** Reuniones diarias de Scrum se organizan para realizar un seguimiento de los progresos del Equipo Scrum de forma continua y también sirven como reuniones de planificación. Se pregunta qué se ha hecho desde la última reunión y lo que hay que hacer antes de la próxima. También los problemas y otros asuntos se discuten así como las deficiencias o impedimentos en el proceso de desarrollo. El Scrum Master lleva a cabo las reuniones diarias de Scrum. Tiene una duración de no más de 15 minutos y generalmente se realizan con todos los participantes de pie.

7. **Sprint Review meeting:** En el último día de cada Sprint, el Equipo Scrum y el Scrum Master presentan los resultados del Sprint. Los participantes evalúan el incremento del producto. La reunión de revisión puede provocar pedidos nuevos e incluso cambiar la dirección del sistema que se está desarrollando.

Es habitual emplear Scrum como un marco de trabajo ágil para la administración de proyectos. Su uso se está extendiendo cada vez más dentro de la comunidad de Metodologías Ágiles, siendo combinado con otras – como XP – para completar sus carencias. [15]

1.2.2.3 Crystal Methodologies

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. Han sido desarrolladas por Alistair Cockburn.

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. Estas políticas dependerán del tamaño del equipo, estableciéndose una clasificación por colores, por ejemplo Crystal Clear (3 a 8 miembros) y Crystal Orange (25 a 50 miembros).

Crystal está centrado en:

1. Las personas.
2. La interacción.
3. La comunidad.
4. Las habilidades.
5. Los talentos.
6. La comunicación.

Una cuestión interesante que surge del análisis de la serie Crystal es el pragmatismo con que se optimiza el proceso. Las personas involucradas escogen aquellos principios que les resultan efectivos y mediante la aplicación de la metodología en diversos proyectos agregan o remueven principios en base al consenso grupal del equipo de

desarrollo. [15]

Para Cockburn los procesos pese a ser importantes son secundarios, respecto a los principios en que está orientado este conjunto de metodologías, ya que la propia complejidad de la gestión de los equipos para obtener de los mismos el máximo rendimiento, teniendo en cuenta que están formados por un conjunto diverso de personalidades, experiencias, talentos y habilidades, es de por sí el principal problema que hay que resolver en el desarrollo de software. [16]

Prácticas:

1. **Staging:** Estadificación, incluye la planificación del siguiente incremento del sistema. Se deben programar para producir una versión cada tres o cuatro meses como máximo; Cockburn recomienda de uno a tres. El equipo selecciona los requisitos a ser implementados en el incremento y horarios preferibles.
2. **Revision and review:** Cada incremento incluye varias iteraciones. Cada iteración incluye las siguientes actividades: construcción, demostración y revisión de los objetivos del incremento.
3. **Monitoring:** El progreso es monitoreado en relación con las prestaciones del equipo durante el proceso de desarrollo con respecto a su progreso y la estabilidad. El progreso se mide por etapas (inicio, revisión 1, revisión 2, prueba, entrega) y las etapas de estabilidad (altamente fluctuantes, fluctuante y lo suficientemente estable como para revisar).
4. **Parallelism and flux:** Una vez que el seguimiento de la estabilidad da el resultado "lo suficientemente estable como para revisar" las prestaciones de la siguiente tarea pueden comenzar. En Crystal Orange, esto significa que los múltiples equipos puede proceder con paralelismo. Para asegurar esto, los equipos de monitoreo y de arquitectura deben revisar sus planes de trabajo, la estabilidad y la sincronización.
5. **Holistic diversity strategy:** Crystal Orange incluye un método denominado estrategia de diversidad global para la división de grandes equipos funcionales en grupos multifuncionales. La idea central de esto es incluir múltiples

especialidades en un solo equipo. La estrategia de diversidad global permite también la formación de equipos pequeños con un know-how necesario, y también considera temas como la localización de los equipos, la comunicación, la documentación y la coordinación de varios equipos.

6. **Methodology-tuning technique:** La técnica de la metodología de ajuste es una de las técnicas básicas de Crystal Clear y Orange. Utiliza entrevistas y talleres del proyecto de equipo para la elaboración de una metodología específica para cada proyecto. Una de las ideas centrales del desarrollo incremental es permitir la fijación o la mejora del proceso de desarrollo. En cada incremento, el proyecto puede aprender y utilizar los conocimientos que ha adquirido para el desarrollo del proceso para el siguiente incremento.
7. **User viewings:** Dos revisiones de los usuarios se sugieren para Crystal Clear por cada liberación. En Crystal Orange, los comentarios de los usuarios se deben realizar en tres ocasiones para cada incremento.
8. **Reflection workshops:** Tanto Crystal Clear y Orange incluyen una regla que un equipo debe mantener pre y post-incremento de talleres de reflexión (con la recomendación de talleres de reflexión también a mediados de incremento).

Crystal Clear y Crystal Orange no definen las prácticas o técnicas específicas para ser utilizadas por los miembros del proyecto en sus tareas de desarrollo de software. La adopción de prácticas de otras metodologías como XP y Scrum está permitido en Cristal para reemplazar algunas de sus propias prácticas, tales como talleres de reflexión, por ejemplo. [17]

1.2.2.4 Feature-Driven Development (FDD)

Define un proceso iterativo que consta de cinco pasos. Las iteraciones son cortas (hasta 2 semanas). Se centra en las fases de diseño e implementación del sistema partiendo de una lista de características que debe reunir el software. Sus impulsores son Jeff De Luca y Peter Coad.

FDD posee una jerarquía de features, siendo el eslabón superior el de feature set que agrupa un conjunto de features relacionadas con aspectos en común del negocio. Por último, establece el major feature set como el más alto nivel de agrupación de

funcionalidad que abarca diferentes feature sets que contribuyen a proveer valor al cliente en relación a un subdominio dentro del dominio completo de la aplicación. Una de las ventajas de centrarse en los features del software es el poder formar un vocabulario común que fomente que los desarrolladores tengan un diálogo fluido con los clientes, desarrollando entre ambos un modelo común del negocio. [15]

FDD consiste de un conjunto de "mejores prácticas" y los desarrolladores del método argumentan que a pesar de las prácticas seleccionadas no son nuevos, la mezcla específica de estos ingredientes hacen que los cinco procesos de FDD sean únicos para cada caso.

Prácticas:

1. **Domain Object Modeling:** exploración y explicación del dominio del problema. Los resultados en un marco donde los rasgos se añaden.
2. **Developing by Feature:** El desarrollo y seguimiento de los progresos a través de una lista de pequeñas funcionalidades descompuestas y valoradas por el cliente.
3. **Individual Class (Code) Ownership:** Cada clase tiene una sola persona nominada para ser la responsable de la integridad, consistencia, rendimiento y conceptualizar la clase.
4. **Feature Teams:** Se refiere a los equipos pequeños y dinámicamente formados.
5. **Inspection:** Se refiere al uso de los más conocidos mecanismos de detección de errores.
6. **Regular Builds:** Se refiere al aseguramiento de que siempre exista un sistema en funcionamiento, demostrable y disponible. Versiones regulares para formar la línea base a la que se añaden nuevas funciones.
7. **Configuration Management:** Permite la identificación y el seguimiento histórico de las últimas versiones de cada archivo código fuente.
8. **Progress reporting:** El progreso se reporta sobre la base del trabajo completado en todos los niveles organizativos necesarios.

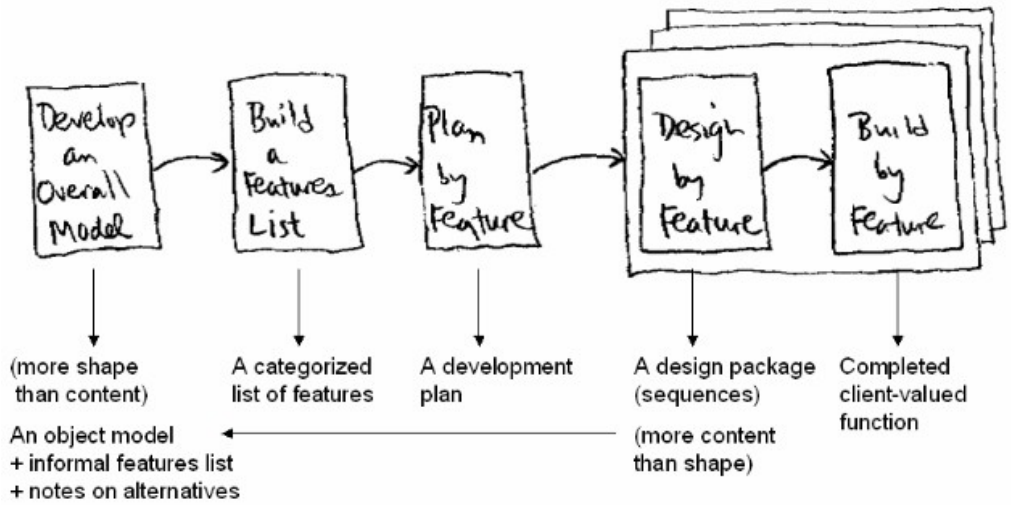

Ilustración 7: Los cinco procesos dentro de FDD. (Coad, 2000)

1.2.2.5 SXP

SXP, es un híbrido cubano de metodologías ágiles, que ofrece una estrategia tecnológica, a partir de la introducción de procedimientos ágiles que permitan actualizar los procesos de software para el mejoramiento de la actividad productiva, fomentando el desarrollo de la creatividad, aumentando el nivel de preocupación y responsabilidad de los miembros del equipo; además ayuda al líder del proyecto a tener un mejor control del mismo. Consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar el éxito del proyecto. Basada completamente en los valores y principios de las metodologías ágiles expuestos en el Manifiesto Ágil. Como método de estimación se utiliza la opinión de expertos y consta con métricas o indicadores para lograr una eficiente calidad. [1]

Fases:

1. **Planificación-Definición:** donde se establece la visión, se fijan las expectativas y se realiza el aseguramiento del financiamiento del proyecto.
2. **Desarrollo:** donde se realiza la implementación del sistema hasta que esté listo para ser entregado.

3. **Entrega:** puesta en marcha.
4. **Mantenimiento:** donde se realiza el soporte para el cliente.

SXP está especialmente indicada para proyectos de pequeños equipos de trabajo, rápido cambio de requisitos o requisitos imprecisos, muy cambiantes, donde existe un alto riesgo técnico y se orienta a una entrega rápida de resultados y una alta flexibilidad. Ayuda a que trabajen todos juntos, en la misma dirección, con un objetivo claro, y permite además seguir de forma clara el avance del equipo de desarrollo por parte del cliente, de forma que los jefes pueden ver día a día cómo progresa el trabajo. [1]

Entre los resultados obtenidos, según sus autores G. Peñalver, A. Meneses y S. García, se encuentran:

1. La metodología establece el uso de sistemas automatizados para la generación de algunos artefactos. Y además los recomienda de manera explícita.
2. Contar con un expediente le permitió sobrevivir, llegar a usarse. Establecerse en el entorno con apoyo legal.
3. Muy usada en proyectos que no utilizan paradigma orientado a objetos. Fundamentalmente proyectos de Software Libre (SWL) que trabajan con ficheros Bash o C.
4. Positiva para el uso de estudiantes que aprenden ingeniería. En la práctica los estudiantes de años avanzados en la carrera Ingeniería en Ciencias Informáticas son los únicos que pueden desempeñar el rol de analistas o diseñadores de un software, lo que hace muy improductiva la fuerza joven, que en ocasiones tiene buenas ideas y energía para desarrollarlas, e incluso muy buena preparación en uno o dos lenguajes de programación.
5. SXP cuenta con artefactos muy bien definidos para períodos de investigación. En el desarrollo de software libre, para no reinventar la rueda, se hace necesario investigar los proyectos que han intentado hacer lo mismo. En un banco de software disponible de unos 200 000 proyectos, los períodos de investigación suelen ser de unos 3 ó 4 meses. Posterior a ello el equipo comienza siempre a desarrollar partiendo de código ya desarrollado y que fue

encontrado durante la investigación. Documentar estos 4 meses de investigación – en pocos artefactos - suele ser productivo para que otros equipos de desarrollo tomen decisiones en menos tiempo. También permite a desarrolladores en entornos académicos obtener publicaciones.

6. SXP ha sido excelente para equipos pequeños, siempre ha sido recomendado para menos de 20 personas.
7. La documentación siempre tiene un día o una semana de retraso con respecto al código. Siempre hay más código que documentación. En el entorno UCI con metodología RUP suele haber mucho papel y poco código. Lamentablemente el proceso de revisión y auditorías no tiene un acápite para revisar el código, o sea, si usted ha dicho que usa una metodología ágil lo más conveniente sería revisar las Historias de Usuario (HU) que debe haber implementado según el cronograma y hacer al menos pruebas de caja negra a la aplicación para evaluar la veracidad de la documentación.
8. El tiempo en empezar a desarrollar ha sido bajado hasta 14 días, una cifra extraordinaria para el entorno UCI pues los proyectos por lo general pasan más de 2 meses y hasta 10 en definición. SXP ha demostrado que puede tener 3 o 4 versiones BETAs con funcionalidades relevantes, y varias versiones ALFAs intermedias a estas.
9. No ha sido necesario abandonar los diseños anteriores de los analistas. En el criterio particular de los investigadores esto podría tratarse de lo siguiente, o sea la explicación: los analistas tienen poca experiencia de desarrollo. Hacen sus análisis y después en tiempo de implementación, los desarrolladores deben cambiar más del 30% de lo que ya se ha escrito. Esto constituye una demora al proyecto, y un fracaso a la etapa de análisis.
10. SXP generó su propia documentación basada en 8 tesis, las cuales están en el repositorio de SXP para su consulta todo el tiempo.
11. Las pruebas en SXP han sido por lo general satisfactorias.
12. Estableciendo un sistema de trabajo colaborativo SXP tiene una plantilla de Arquitectura que ha pasado por múltiples manos. SXP también tiene asociado

el concepto de visión de Arquitectura: Si su equipo no sabe como hacer este sistema ¡píntalo! Luego programación chatarra... e itera todo eso cada vez que sientas que estás en un escalón superior.

1.2.3 Metodologías Ágiles vs Metodologías Tradicionales

Cada metodología de desarrollo de software, ya sea tradicional o ágil, posee ventajas y desventajas. Se comparte el criterio de que las metodologías tradicionales como RUP no son viables en proyectos donde la satisfacción del cliente a corto tiempo es importante; y en la que dicha satisfacción se logra entregándole una versión ejecutable pero que evidentemente no posee todas las funcionalidades que solicitó.

Otro aspecto es la cantidad de documentos a realizar en las tradicionales, algo que atrasaría una entrega temprana del producto. En [1] sus autores escriben sobre RUP "...no se consideraba viable puesto que orientaba todo el trabajo ingenieril a una sobre documentación del proyecto que hacía poco gestionable el mismo, además que hacía al grupo dependiente en cierta forma de herramientas CASE de corte propietario como el Rational Rose o Visual Paradigm, o en la búsqueda de las herramientas alternativas, se debían utilizar muchas herramientas que cada una por su parte realizaba lo que las privativas hacen en una sola, siendo esto un gran problema, no tanto por el tema de la comodidad, sino desde el punto de vista de la trazabilidad que tienen todos los artefactos que se generan en dicho proceso, provocando en ocasiones inconsistencias entre los artefactos y una sobre dosis de Gestión de la Configuración y Cambios que ya incidía negativamente en el desarrollo y mantenibilidad de los proyectos y productos respectivamente."

En el **Anexo D** se recoge esquemáticamente las principales diferencias de las metodologías ágiles con respecto a las tradicionales según [18]

Algo positivo de las metodologías ágiles, es que poseen ciertas propiedades que las hacen totalmente aplicables al dominio del software en los móviles. En [4] [19] se realiza una comparativa entre las características básicas o bases (home ground) ágiles y los rasgos observados en el desarrollo de software para dispositivos móviles.

Características ágiles	Motivación lógica	En el caso del desarrollo para plataformas móviles.
Alta volatilidad del entorno	Debido a la alta frecuencia en el cambio que sufren los requerimientos, se tendrá menos necesidad de diseño y planificación inicial y mayor necesidad de desarrollos incrementales e iterativos.	Alta incertidumbre, entornos dinámicos, cientos de nuevos terminales cada año.
Equipos de desarrollo pequeños	Capacidad de reacción más rápida, trabajo basado en la compartición de la información, menos documentación.	La mayor parte de los proyectos de desarrollo de software para plataformas móviles se lleva a cabo en microempresas y PyME.
Cliente identificable	Desaparecen los malentendidos.	Potencialmente, hay un número ilimitado de usuarios finales, pero los clientes son fáciles de identificar.
Entornos de desarrollo orientados a objetos	La mayoría de las herramientas de desarrollo ágil existen bajo plataformas orientadas a objetos.	Por ejemplo, Java y C++ se usan, algunos problemas en herramientas como refactorizaciones o primeros tests.
Software crítico no asegurado	Los fallos no causan gran impacto, como la pérdida de vidas. Se puede buscar mayor agilidad en el desarrollo.	La mayoría del software es para entretenimiento. Los terminales no son fiables.
Software a nivel de aplicación	Sistemas embebidos grandes requieren comunicación exhaustiva y mecanismos de verificación.	Mientras los sistemas móviles son complejos y altamente dependientes, las aplicaciones son muy autónomas.
Sistemas pequeños	Menos necesidad de diseño inicial.	Las aplicaciones, aunque variables en tamaño, no suelen superar las 10.000 líneas de código.
Ciclos de desarrollo cortos	Propósito de realimentación rápida.	Periodos de desarrollo de 1 a 6 meses.

Tabla 1: Comparativa entre las características básicas o bases (home ground) ágiles y los rasgos observados en el desarrollo de software móvil.

1.2.4 Metodologías utilizadas en el desarrollo de aplicaciones para móviles

En la literatura se pueden encontrar algunas propuestas de metodologías para el desarrollo de aplicaciones para dispositivos móviles, entre ellas se encuentran Dynamic Channels, Mobile-D y un modelo híbrido. Si bien no son populares, tienen algunos elementos interesantes en comparación con las descritas anteriormente.

1.2.4.1 Dynamic Channels

Sus autores manifiestan que la metodología se deriva de OMT (Object Modeling Technique), aumentada con casos de uso. También diseñan formas para capturar información específica del canal, como la programación y la notificación. Utilizan la notación UML para describir los modelos de canal que se desarrollan. Aunque la metodología cubre todas las fases del ciclo de vida de desarrollo, la discusión se centra en la fase de análisis, donde los usuarios construyen modelos de solicitud utilizando el concepto de canal dinámico.

La metodología comienza, como en OMT, mediante la construcción de un modelo del dominio de aplicación. A continuación, se añaden modelos que describen cómo el diseño evoluciona. La metodología contiene las siguientes etapas (o fases): Análisis, Diseño de objetos e Implementación. [20]

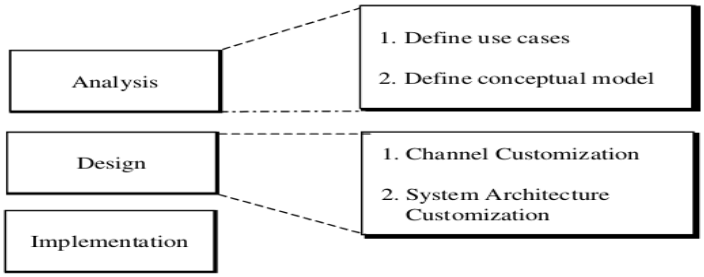

Ilustración 8: Actividades de desarrollo de Dynamic Channel (Alfonso, 1998)

El canal es primeramente descrito durante la fase de análisis, basado en la elaboración de casos de uso y personalizada mediante un meta-modelo para el canal de información. El modelo resultante es expandido a un modelo de diseño que representa la solución prevista. Por último, es implementado haciendo uso los componentes de software pertenecientes al marco de trabajo de la empresa donde se desarrolla. [20]

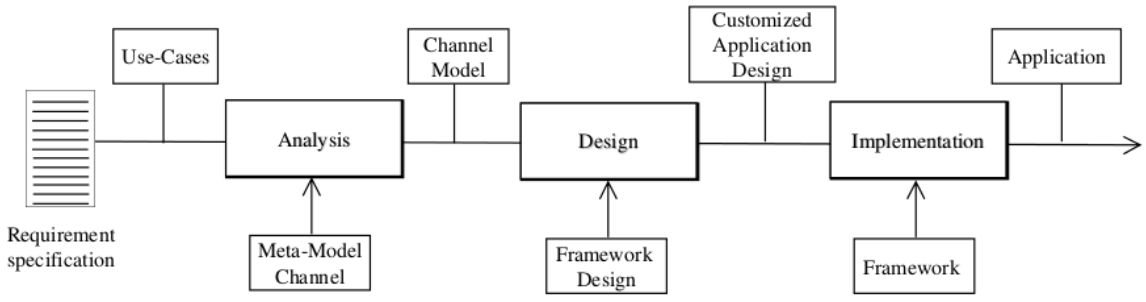

Ilustración 9: Metodología de desarrollo Dynamic Channel: Entradas y artefactos (Alfonso, 1998)

1.2.4.2 Mobile-D

La aproximación de Mobile-D se ha apoyado en muchas otras soluciones bien conocidas y consolidadas: eXtreme Programming (XP), Crystal methodologies y Rational Unified Process (RUP). Los principios de programación extrema se han reutilizado en lo que se refiere a las prácticas de desarrollo, las metodologías Crystal proporcionaron un input muy valiosos en términos de la escalabilidad de los métodos y el RUP es la base para el diseño completo del ciclo de vida. [4]

Según Abrahamsson el proceso de móvil-D debe ser utilizado por un equipo de más de diez, trabajando para una entrega del producto dentro de diez semanas. [21]

Algunas prácticas de Mobile-D:

1. Siguiendo una Línea de Arquitectura.
2. Desarrollo orientado a pruebas. (TDD, Test-Driven Development).
3. Integración continua.
4. Programación en pares.
5. Métricas.
6. Cliente fuera.
7. Centrado en el usuario.

La Línea de Arquitectura en la metodología es una nueva adición a las prácticas ágiles ya establecidas. Una línea de la arquitectura se utiliza para capturar el conocimiento de una organización de soluciones arquitectónicas, tanto de fuentes internas y externas, y usar estas soluciones cuando sea necesario.

Mobile-D se compone de cinco fases: Exploración, Inicio, Producción, Estabilización y Pruebas y Corrección. Cada una de estas fases tiene una serie de etapas, tareas asociadas y prácticas.

En la primera fase, Exploración, el equipo de desarrollo debe generar un plan y establecer las características del proyecto. Esto se hace en tres etapas: establecimiento de las partes interesadas, la definición del alcance y el establecimiento del proyecto. Las tareas asociadas a esta fase incluyen el establecimiento del cliente (los clientes que toman parte activa en el proceso de desarrollo), la planificación del

proyecto inicial y la recolección de requisitos, y el establecimiento del proceso.

En la siguiente fase, Inicio, el equipo de desarrollo y todas las partes activas deben entender el producto que se desea desarrollar y se dedican a la preparación de los recursos fundamentales necesarios para las actividades de producción, tales como los recursos físicos, tecnológicos y de comunicaciones. Esta fase se divide en tres etapas: la puesta en marcha del proyecto, la planificación inicial y el día del juicio.

La fase de Producción comprende principalmente las actividades de implementación. Al final de esta fase, la mayor parte de la aplicación debe estar completa. Esta fase se divide en días de planificación, días laborables, y los días de liberación. Los días de planificación tienen por objeto mejorar el proceso de desarrollo, priorizar y analizar las necesidades, la planificación de los contenidos de la iteración, y la creación de pruebas de aceptación que se ejecutarán más tarde en los días de salida.

En días laborables, lo más importante es el Test-Driven Development (TDD), práctica que se utiliza para implementar funcionalidades, de acuerdo con el plan preestablecido para la iteración actual. Utilizando TDD junto con la integración continua, los desarrolladores crear pruebas unitarias, escriben código que pase las pruebas, e integran el nuevo código con la versión existente del producto. Por último, en los días de salida de una versión del sistema se validada a través de las pruebas de aceptación.

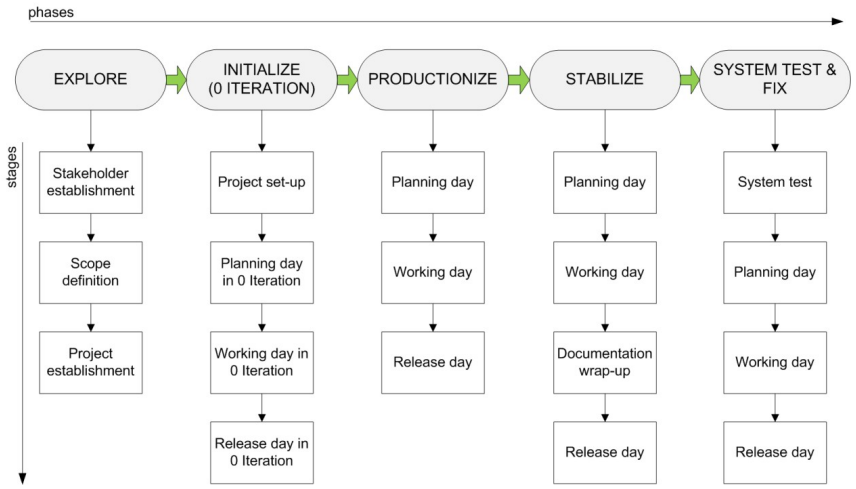

Ilustración 10: Fases Mobile-D

Las dos últimas fases, Estabilización y Prueba, y Corrección, se utilizan para la finalización y comprobación del producto respectivamente. Ellos comprenden las etapas similares a la fase de Producción, con algunas modificaciones para dar cabida a la creación de documentación y pruebas del sistema.

1.2.4.3 Un modelo híbrido para el desarrollo ágil

La aproximación metodológica de Rahimian y Ramsin se apoya en una combinación del desarrollo adaptativo de software (Adaptive Software Development, ASD) y el diseño de nuevos productos (New Product Development, NPD). Esto supone una decisión crítica para decantarse más del lado del desarrollo de productos que del lado de la gestión de proyectos, lo cual quiere decir que una de las característica más sensibles, desde el punto de vista metodológico, para la consolidación de una metodología propia de un entorno móvil, es la presión de los plazos para llegar al mercado, un mercado volátil y altamente dinámico. [4]

Ha sido elaborado mediante un proceso iterativo de diseño híbrido de metodologías. En [19] se detallan las iteraciones siendo la cuarta la última.

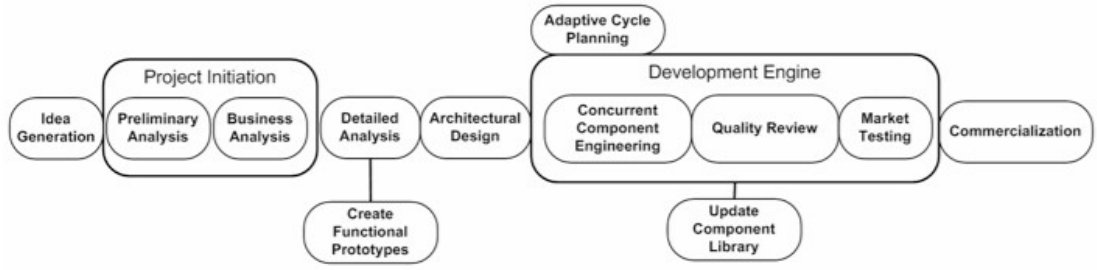

Ilustración 11:Diseño híbrido para el desarrollo ágil (Rahimian y Ramsin, 2008)

Sus autores comienzan con el típico ciclo de vida tradicional (análisis, diseño, implementación, prueba y transición) y mediante cuatro iteraciones logran realizar un diseño donde sus principales características son las siguientes:

1. Introducen una pequeña fase de generación de ideas.
2. Otorgan una gran importancia a la mitigación de riesgos en el desarrollo.
3. Se añaden elementos de prototipado.

4. Introducen diseño basado en arquitectura.
5. Introducen conceptos de desarrollo orientado a pruebas (Test-Driven Development, TDD)
6. Añaden una prueba de mercado antes de llegar a la fase Comercialización.
7. Introducen una fase llamada Comercialización.

Conclusiones del capítulo

Teniendo en cuenta el exponencial aumento del desarrollo de aplicaciones para dispositivos móviles y además a las cambiantes tecnologías se hace evidente una entrega lo más temprana posible; por lo que sin duda las metodología ágiles se adaptan más que las tradicionales en este tipo de proyecto.

Las metodologías tradicionales asumen que el cliente no puede desconocer algún requerimiento, debe explicarlos a los desarrolladores para que luego realicen una guía minuciosa para dar cumplimiento a cada uno de ellos en el tiempo establecido. En cambio haciendo uso de las ágiles el proyecto puede comenzar sin que el cliente tenga claro todas las funcionalidades. En la mayoría de los proyectos que desarrollan aplicaciones para móviles no se poseen todos las funcionalidades en un inicio al ser muy cambiantes, ya sea por ideas del equipo o por algún cambio de tecnología. Por lo que es evidente el uso de metodologías ágiles.

Es notable que las ágiles tienen como objetivo la satisfacción del cliente en todo momento, la adaptabilidad a los cambios, la entrega de productos rápidamente y dentro del presupuesto estipulado. Han demostrado sus beneficios en términos de salida rápida al mercado, adaptabilidad a la evolución de los requisitos y satisfacción del cliente.

Además se puede adicionar que en los últimos años las organizaciones que desarrollan software se han beneficiado de las metodologías ágiles tales como XP y Scrum; pues ya no son pocos los artículos publicados que reflejan el éxito obtenido por algunos proyectos que han adoptado metodología ágiles; por lo que muchas organizaciones aspiran a utilizarlas.

Dentro de las ágiles, la familia Crystal, propuesta por Alistair Cockburn poseen un bajo

nivel de disciplina, característica que no se desea en las metodologías empleadas por estudiantes y desarrolladores de la UCI. A diferencia de la familia Crystal, XP posee un alto nivel de disciplina, algo que sin duda se tendrá en cuenta.

En el desarrollo de aplicaciones para dispositivos móviles es necesario un alto nivel de imaginación y creatividad. Logrando tales indicadores existirá una mayor probabilidad que el producto final tenga éxito. Resulta evidente integrar al equipo personas que realmente deseen participar y ser protagonistas del proceso. Conociendo que cada estudiante en el ciclo productivo (segundo y último ciclo de formación en la Facultad Regional Granma perteneciente a la Universidad de las Ciencias Informáticas) es asignado a un proyecto y desde el mismo es formado como Ingeniero en Ciencias Informáticas, resulta interesante lograr aplicar una metodología de desarrollo capaz de ayudar en este objetivo social e intelectual.

Capítulo II: Propuesta de metodología de desarrollo de software

Introducción

Existen algunas características básicas de una metodología ideal para el proceso de desarrollo de software para plataformas móviles. A continuación se relacionan algunas de ellas: [19]

1. **Agilidad:** Las metodologías ágiles mejoran la flexibilidad del desarrollo y la productividad, proveyendo métodos que se adaptan a los cambios y que aprenden de la experiencia. Características específicas de los entornos móviles deben ser: procesos iterativos e incrementales, desarrollo conducido por pruebas, procesos adaptativos, hablar con el cliente continuamente, desarrolladores altamente cualificados, asegurar la calidad, revisiones continuas del proceso, priorización de los requerimientos y bajo time-to-market.
2. **Conciencia de mercado:** El mercado actual está orientado hacia los productos software por lo que un proceso de desarrollo móvil debería enfocarse al desarrollo del producto y no del proyecto. Consecuentemente, los procesos deben enfocarse al nicho del negocio. Toda esta información mitigará las dudas y los riesgos. Procesos orientados al mercado seguirán una planificación bastante estricta a lo que se refiere a requerimientos de time-to-market. Antes los procesos estaban orientados a las actividades técnicas ahora tienen que hacer un balanceo entre las actividades orientadas a mercado y las técnicas.
3. **Soporte a toda la línea de producción software:** Se refiere al conjunto de sistemas intensivos de software compartiendo un conjunto de características comunes que satisfacen las necesidades de un segmento particular del mercado y que son desarrolladas con una serie de valores centrales en una forma predeterminada. Esto hace que el ciclo de vida sea más corto, pudiendo desarrollar una familia de productos software móviles en un solo intento, reduciendo costes. Debe tener especial importancia la línea de producción para la mejora de la calidad del software.
4. **Desarrollo basado en arquitectura:** La eficiencia de la línea de producción de

software depende del desarrollo de una plataforma común, por lo que la necesidad de una arquitectura genérica para una clase de productos es esencial, pudiendo reconfigurarse de forma específica para cada componente o producto determinado.

5. **Soporte para reusabilidad:** El desarrollo basado en componentes y el basado en capas ahorra costes de desarrollo, agiliza la entrega del producto y hace el software menos propenso a errores ya que los componentes no deben ser hechos desde cero cada vez.

6. **Inclusión de sesiones de revisión y de aprendizaje:** La metodología debería incorporar sesiones de revisión en todo el proceso para asegurar el análisis del producto y sesiones de aprendizaje después de la entrega de cada producto para que la experiencia sea analizada y registrada, y así la abstracción del conocimiento obtenido realimente a todo el equipo.

7. **Especificación temprana de la arquitectura física:** La arquitectura física debe ser elaborada en las etapas tempranas del desarrollo software gracias a que un alto número de riesgos técnicos son inherentes a las limitaciones de los dispositivos móviles y las diferencias en la implementación pueden ser obtenidas de características básicas. El uso de un prototipo mitigaría dichos riesgos técnicos.

Teniendo en cuenta estas características y conociendo de que si se sostiene una fuerte disciplina sin agilidad, el resultado será burocrático y se estancará, y si se posee un proceso ágil sin disciplina se desarrollará sin restricciones y baja calidad, se decidió optar por la mezcla de ambas.

2.1 Características de la metodología propuesta

Para la descripción de la metodología que se propone, a la que se ha nombrado como SXP-J2ME, se ha seguido la propuesta de Alistair Cockburn. Ella desglosa una metodología en 10 elementos como mínimo. Dichos elementos se describen a continuación: [22] [23]

1. **Roles**: las descripciones a los trabajos que se solicitan en las búsquedas laborales cuando se necesitan tomar gente. Ejemplo: project manager,

arquitecto, escritor, tester.

2. **Destrezas**: las destrezas que presupone poseen los recursos que llevan a cabo el proyecto. Ejemplo: social, proactivo, hábil en el manejo de herramientas.
3. **Entregables**: lo que se quiere que cada persona o equipo entregue a otra persona o equipo. Ejemplo: casos de uso, especificaciones de diseño, documentación de framework, diagramas de secuencia.
4. **Actividades**: las actividades e hitos de los diferentes equipos, cómo se integran para producir los entregables finales.
5. **Valores**: los valores del equipo y de la propia metodología.
6. **Equipos**: la forma en que se agrupan a las personas.
7. **Asignación de Tareas**: cómo son asignadas las tareas a los múltiples roles.
8. **Técnicas**: las técnicas que se espera que las personas utilicen en su trabajo.
9. **Herramientas**: las herramientas que las personas usan a diario, ya sea dentro de una técnica o para producir un entregable de acuerdo al estándar.
10. **Estándares**: lo que está o no permitido en el diseño y los entregables. Esto incluye notación, convenciones del proyecto y cuestiones de calidad.

2.1.1 Roles definidos

Equipo del Proyecto (Team): Es el equipo del proyecto que tiene la autoridad para decidir cómo organizarse para cumplir con los objetivos de un Sprint.

Típicamente es un equipo de 2 a 8 personas, cada una especializada en algún elemento que conforma los objetivos a cumplir, por ejemplo: desarrollador, diseñador, etc. La membrecía solo puede cambiar entre Sprints (no durante).

Tareas fundamentales:

1. Estimar esfuerzo.
2. Crear la reserva del Sprint.
3. Revisar la Lista de Reserva del Producto.
4. Sugerir obstáculos que deban ser removidos para cumplir con los ítems que

aparecen.

Jefe de Proyecto (Project Manager): Es un rol de administración que debe asegurar que el proyecto se está llevando a cabo de acuerdo con la metodología de desarrollo y que todo funciona según lo planeado.

Tareas fundamentales:

1. Remover impedimentos en el desarrollo de los productos.
2. Reducir riegos en el desarrollo de los productos.
3. Fomenta la cohesión del grupo.
4. Lleva a cabo actividades destinadas a eliminar fricciones.

Jefe de Producto (Product Management): Es el responsable de tomar las decisiones finales, acerca de estándares y convenciones a seguir durante el desarrollo de un producto determinado. Participa en la selección de objetivos y requerimientos.

Tareas fundamentales:

1. Realiza la planificación del producto, a lo largo de todo el ciclo de vida, incluida la planificación en detalle de cada sprint.
2. Asegura que se consigan los objetivos propuestos en la reunión de planificación del Sprint.
3. Controla el progreso del desarrollo del producto.
4. Trabaja junto con el Jefe de Proyecto en la reducción de la Lista de Reserva del Producto.

Arquitecto (Architect): Se vincula directamente con los analistas y diseñadores debido a que su trabajo tiene que ver con la estructura y el diseño en general de las aplicaciones.

Tareas fundamentales:

1. Definición de la arquitectura empleada en el proyecto.
2. Definirá los lineamientos generales del diseño y la implementación.
3. Deberá construir cualquier prototipo necesario para probar aspectos riesgosos

desde el punto de vista técnico en el proyecto.

4. Mantiene al equipo actualizado en cuanto a los nuevos componentes subidos al repositorio de componentes del proyecto.

Visionario (Visionary): Es la persona que luego de expresar una idea y solicitar que sea desarrollada es aprobada por el Jefe de Proyecto junto con otros especialistas.

Tareas fundamentales:

1. Propone al Jefe de Proyecto un posible producto a desarrollar.
2. Realiza la primera versión de la factibilidad del producto a desarrollar.
3. Otorga la prioridad de cada una de las funcionalidades a desarrollar.
4. Junto con el Encargado de Prueba genera las pruebas funcionales a partir de los requerimientos.

Analista (Analyst): Encargado de realizar el análisis correspondiente al producto que se desarrolla.

Tareas fundamentales:

1. Obtienen las funcionalidades del producto a desarrollar.
2. Realiza la especificación de las funcionalidades.
3. Escribe las Historias de Usuario.
4. Junto con el Visionario asigna la prioridad a las Historias de Usuario.

Diseñador (Designer): Encargado del diseño del sistema mediante tarjetas CRC; así como el de los prototipos de interfaces.

Tareas fundamentales:

1. Responsable de realizar el diseño de las metáforas.

Desarrollador (Developer): Es el encargado de producir el código y escribir las pruebas unitarias. Debe comunicarse frecuentemente con los restantes miembros del equipo.

Tareas fundamentales:

1. Es el responsable de la codificación de los componentes a desarrollar.

2. Debe crear y ejecutar las pruebas unitarias realizadas sobre el código desarrollado.
3. Es responsable de documentar, actualizar ante cambios y mantener bajo control de configuración las clases que ha desarrollado.

Encargado de Pruebas (Tester): Es el encargado de ayudar al Visionario a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.

Tareas fundamentales:

1. Es el responsable de generar las pruebas funcionales a partir de los requerimientos extraídos por los analistas.

Consultor (Consultant): Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan sugerir problemas, además aporta ideas y experiencias para el beneficio del sistema en desarrollo. Esta es una especialización menos activa, quien la ejecuta funciona por un corto período de tiempo. Está más orientada a roles de desarrollo, y sus ejecutores pueden trabajar en otros roles (en otro equipo) durante el desarrollo del software.

Tareas fundamentales:

1. Imparte conferencias, charlas, etc, sobre algún campo específico en la actualidad o experiencia obtenida en otros proyectos.
2. Expresa su criterio de alguna actividad en específico o sobre un artefacto determinado.

2.1.2 Destrezas

Jefe de Proyecto: Debe poseer aptitudes en comunicación para poder capturar y entender la visión generada por cualquier tipo de persona. Debe saber escuchar y ser optimista en todo momento.

Jefe de Producto: Debe poseer aptitudes en comunicación y ser exigente con el equipo y con sigo mismo.

Arquitecto: Deberá tener una buena formación técnica, contar con experiencia en las

herramientas y técnicas utilizadas. Además deberá tener aptitudes comunicativas para que la arquitectura llegue a todos los miembros del equipo. También deberá ser perseverante en conseguir los hitos técnicos planteados mediante entregables para asegurar el progreso de la construcción.

Visionario: Debe poseer aptitudes en comunicación para poder explicar la visión generada. Además deberá ser una persona decidida.

Analista: Deberá tener amplio conocimiento de técnicas de levantamiento de funcionalidades.

Desarrollador: Deberá tener amplio conocimiento de las herramientas de desarrollo, del lenguaje de programación utilizado y de los aspectos técnicos involucrados.

Encargado de Pruebas: Deberá tener amplio conocimiento de técnicas de pruebas, deberá conocer a fondo la aplicación que probará. También deberá tener conocimientos de programación para trabajar con las pruebas automatizadas.

Consultor: Deberá tener amplio conocimiento y una elevada preparación en los temas que abordará con el equipo de desarrollo.

2.1.3 Entregables

Ficha Técnica: documento donde se escribirá claramente lo que se desea con el producto, así como su posible aceptación o impacto en los usuarios finales. Tendrá un análisis de factibilidad, además contendrá las funcionalidades más importantes del sistema a ser construido y los requerimientos suplementarios que se detecten de forma temprana .

Responsable: Visionario y Jefe de Producto

Lista Reserva de Producto (LRP): documento que posee una lista priorizada que define el trabajo que se va a realizar en el proyecto. Esta lista puede crecer y modificarse a medida que se obtiene más conocimiento acerca del producto (con la restricción de que solo puede cambiarse entre Sprint).

Responsable: Jefe de Producto y Visionario

Historias de Usuario (HU): Documentos donde se especifican las funcionales del producto. Servirán para guiar los demás artefactos y para la construcción de los

componentes del producto. Son escritas en lenguaje natural, no excediendo su tamaño de unas pocas líneas de texto.

Responsable: Analista

Tarjetas CRC: Las clases pueden ser descubiertas usando tarjetas de Colaboración – Responsabilidad – Clases (CRC por sus siglas en inglés). Fueron introducidas por Ward Cunningham y Kent Beck en 1989. Una tarjeta CRC es una tarjeta que muestra: el nombre de la clase y su descripción, la responsabilidad de la clase, conocimiento interno de la clase, servicios brindados por la clase y los colaboradores para las responsabilidades. Un colaborador es una clase cuyos servicios son necesarios para una responsabilidad. **(Ver Anexo K)**

Responsable: Diseñador

Plan de Release: Mediante este documento se realiza una planificación del proyecto. Especifica qué Historias de Usuarios deben ser implementadas para cada uno de los release del sistema. Cada release es conformado por un número de iteraciones (Sprints).

Responsable: Jefe de Producto y Miembros del Proyecto

Lista de Riesgo: se utiliza para capturar los riesgos más relevantes que se presentan en el proyecto, y para poder monitorearlos, asignarles prioridad, impacto y probabilidad de ocurrencia. Sirve para planificar las iteraciones y para tener en vista aquellos riesgos que, por su criticidad, deberán ser mitigados lo más tempranamente posible.

Responsable: Jefe de Proyecto y Jefe de Producto.

Documento de Arquitectura: documento donde se especifican los aspectos técnicos de la solución propuesta por el equipo de desarrollo. Sirve como medio de comunicación entre el Arquitecto y el equipo en relación a cómo deberán ser implementadas las Historias de Usuario.

Responsable: Arquitecto

Informe de Investigación: documentos donde se describen las investigación realizadas, así como resultados obtenidos.

Responsable: Todos

Glosario de Términos: documento que recoge los términos que se relacionan con el producto y la metodología utilizada que pueden causar dudas al equipo. Es necesaria su actualización continua para un mejor entendimiento de todo el proceso de desarrollo de software.

Responsable: Todos

Minuta de Reunión: documento que recoge los acuerdos tomados así como la asistencia a la reunión.

Responsable: Jefe de Producto

Plan de Pruebas: contiene una lista de pruebas a realizar con su fecha de realización correspondiente así como las personas involucradas en las mismas. Divide las pruebas en pruebas de aceptación (pruebas que avala el Visionario) y pruebas de mercado (pruebas realizadas por usuarios finales).

Responsable: Jefe de Producto, Encargado de Pruebas y Visionario

Casos de Prueba: documento que contiene las especificaciones de las pruebas a realizar para la comprobación y validación de las funcionalidades del producto, y de esta forma saber si está apto para ser liberado.

Responsable: Encargado de Pruebas y Visionario

Gestión de Cambio: documento donde se guardan los cambios realizados en el producto una vez terminado el producto y se pasa a la fase donde se mantiene, dándole soporte a los usuarios. Recoge los datos para identificar el nombre de la Historia de Usuario donde se deriva, quién realiza el cambio y el lugar donde ocurre la actualización en el código.

Responsable: Jefe de Producto y Desarrollador

Plan de Capacitación: recoge la planificación del entrenamiento y la preparación necesaria del equipo de desarrollo para lograr los objetivos del proyecto.

Responsable: Jefe de Proyecto y Jefe de Producto

Relación de Herramientas y Componentes: documento que recoge todas las herramientas y componentes utilizadas en el proyecto.

Responsable: Jefe de Proyecto

Manual de Usuario: documento que sirve de ayuda a los usuarios para la interacción con la aplicación.

Responsable: Visionario y Equipos del Proyecto

Documento de Registro: documento legal para el registro del producto.

Responsable: Jefe de Proyecto y Jefe de Producto

2.1.4 Actividades

Las fases permiten mejorar la planificación, ejecución y control del proyecto, se caracterizan por la conclusión y la aprobación de uno o más artefactos entregables que servirán de entrada a la próxima fase o incluso para decidir la continuidad del proyecto. La consecución exitosa de cada fase es indispensable para poder continuar adelante.

SXP-J2ME posee tres fases llamadas Inicio, Elaboración-Construcción y Transición. Las mismas indican el énfasis propuesto por Barry Boehm en relación a la división en dos perspectivas de desarrollo. La primera perspectiva ocurre en las primeras iteraciones del proyecto, cuando se analiza la factibilidad de la ejecución del mismo y se termina con la obtención de la arquitectura y los riesgos más críticos mitigados. La segunda perspectiva ocurre en fases iterativas de construcción en donde el énfasis está en el código fuente generado.

Durante la fase de Inicio el Visionario deberá plasmar los objetivos y elementos que persigue la aplicación, comienza a realizar un estudio de aplicaciones similares existentes así como de los posibles usuarios. Además se realiza un diseño del escenario (diagrama o descripción que refleja la manera que se venderá o divulgará la aplicación). Una vez terminadas las actividades anteriores se realiza un estudio de factibilidad económica de la aplicación propuesta a desarrollar teniendo en cuenta otros beneficios no necesariamente económicos.

En la siguiente fase (Elaboración-Construcción) se describen las funcionalidades a desarrollar mediante Historias de Usuario. Luego se diseñan haciendo uso de las tarjetas CRC y por último son implementadas.

Durante la fase Transición ya deberán existir varias versiones Betas en el mercado con sus respectivos beneficios; no obstante la experiencia dice que existirán

inconvenientes o funcionalidades con resultados no esperados en el accionar de los usuarios por lo que habrá que realizar algunos cambios. En el caso de llegar a una versión estable y con altos niveles de aceptación se procede a registrarlo y se firma el acta de aceptación si procede.

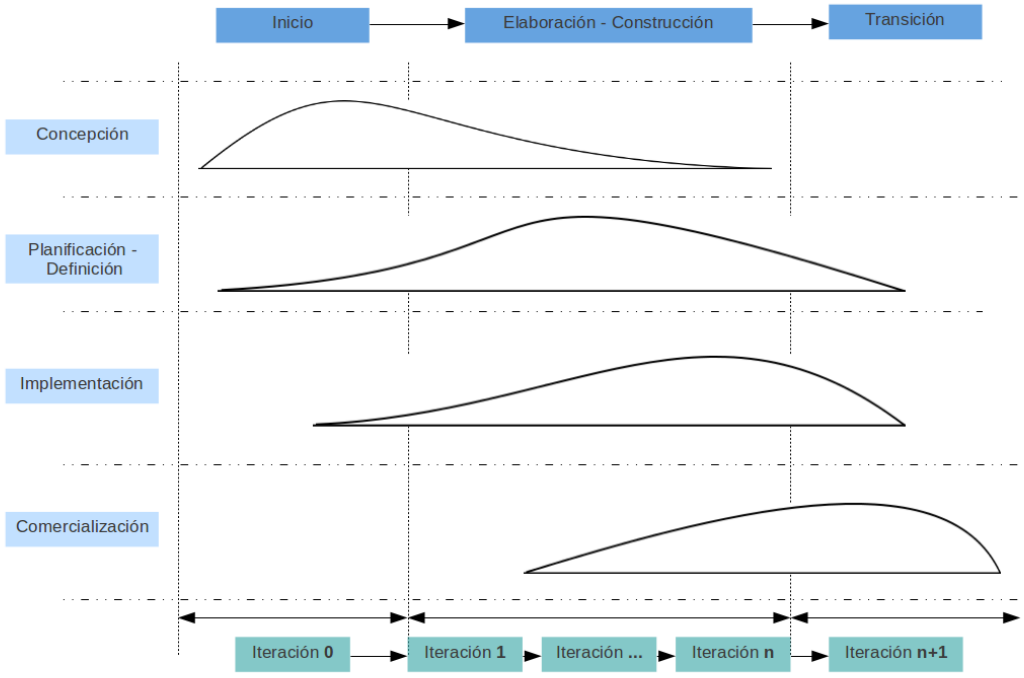

Ilustración 12: Fases y Flujos de SXP-J2ME

Además de las fases se proponen cuatro flujos de trabajo (Concepción, Planificación-Definición, Implementación y Comercialización).

A continuación se describen:

Concepción: Con el surgimiento de una idea por parte de cualquier persona y su evidente deseo de que sea desarrollada, dicha persona toma el rol de Visionario. Se hará un primer bosquejo de las funcionalidades de la aplicación, y la definición de restricciones y cuestiones no funcionales. Todas las tareas realizadas deberán quedar plasmada en un documento llamado “Ficha Técnica”, perteneciente al expediente del proyecto.

Una vez realizado el artefacto antes mencionado, deberá ser presentado a un grupo de especialistas expertos en el dominio y ser criticado; allí se dictará una aprobación o no de la continuidad del proyecto pudiendo surgir nuevas funcionalidades.

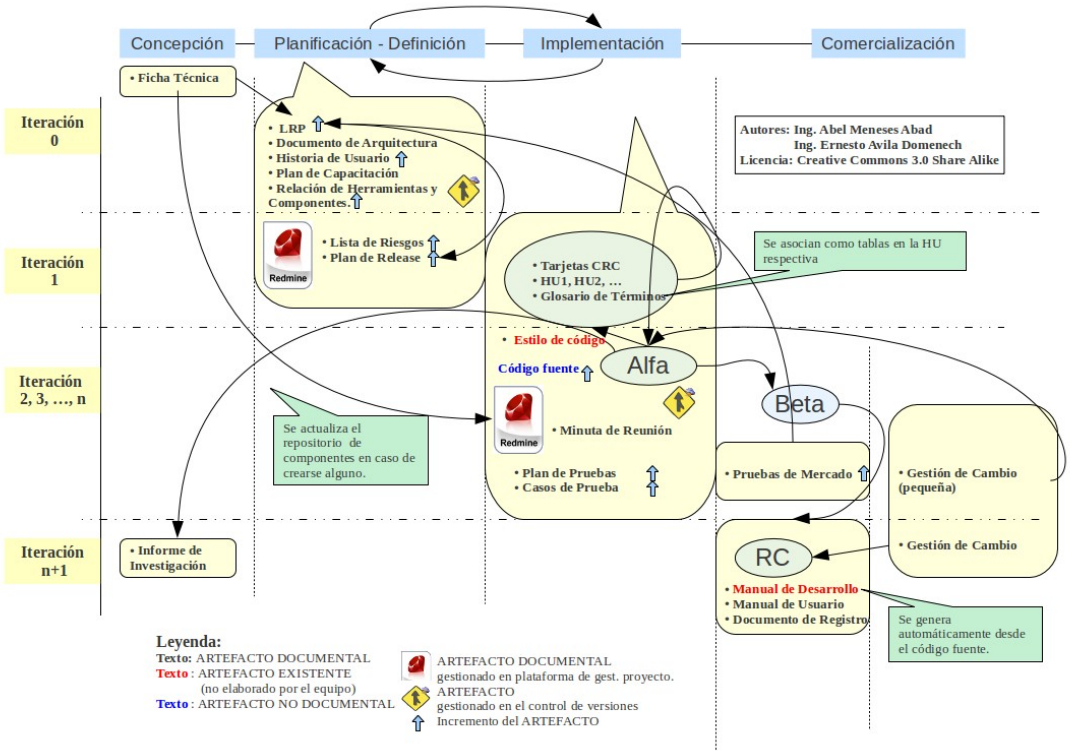

Ilustración 13: Flujos y Artefactos de SXP-J2ME

Planificación-Definición: Aprobada la continuidad del proyecto se fijan las expectativas y se realiza el aseguramiento del proyecto, se asigna la cantidad de estudiantes y especialistas necesarios así como los materiales a utilizar. Luego se realizaría la primera reunión del proyecto donde estaría madurada la idea global de la aplicación a construir y el negocio en que está embebida.

Tomando ideas de FDD una de las actividades a realizar consiste en desarrollar un modelo global, que sugiere un cierto paralelismo con la construcción de la arquitectura del software. Es por ello que surge el artefacto “Documento de Arquitectura”.

Otros documentos del expediente del proyecto que comienzan a ser utilizados son Plan de Capacitación, Relación de Herramientas y Componentes, Lista de Reserva del Producto, Historias de Usuario, Lista de Riesgos y Plan de Release. Pudiendo los dos

últimos ser gestionados desde la plataforma de gestión de proyectos empleada en la línea de productos.

Implementación: Cuenta con tres tareas fundamentales: diseño, generación de código y prueba y optimización del software.

En este flujo se definen las Historias de Usuario a realizar en cada uno de los Sprints mediante el juego de planificación. Tomando idea de Scrum al comienzo de cada Sprint se realiza una reunión llamada “Reunión de Planificación del Sprint” dividida en dos partes. En la primera, denominada “Planificación del Sprint Parte Uno” hay una comunicación entre el Visionario y el Equipo de Proyecto con la facilitación del Jefe de Producto para realizar una estimación del esfuerzo requerido en la implementación de las Historias de Usuario. Esta parte se centra en entender qué quiere el Visionario. Según las reglas de Scrum, al final de esta parte el Visionario se puede marchar aunque debe estar disponible (por ejemplo, por teléfono) durante la siguiente reunión. Sin embargo, es bienvenidos a estar en la misma.

La segunda y última parte llamada “Planificación del Sprint Parte Dos” se centra en la planificación detallada de tareas para saber cómo implementar los elementos que el equipo decide hacer. El Equipo de Proyecto selecciona los elementos de la Lista de Reserva del Producto a las que se comprometen que estará al final del Sprint, comenzando por las de mayor prioridad según el Visionario. Esto es una práctica clave en Scrum y por supuesto en la metodología que se propone: el equipo decide a cuanto trabajo se compromete en vez de serles asignado por el Visionario o Dueño de Producto (en Scrum). Esto hace que el compromiso sea más fiable porque el equipo lo está haciendo basado en su propio análisis y planificación en vez de que venga “hecho” por otros. Aunque el Visionario no tiene control sobre la cantidad de trabajo que el equipo se compromete a hacer, sabe que los elementos que el equipo elija son de los de mayor prioridad –en otras palabras, los elementos que ha clasificado como los más importantes.

Luego de la “Reunión de Planificación del Sprint” son diseñadas las Historias de Usuario mediante tarjetas CRC para luego ser implementadas mediante un desarrollo orientado a pruebas (TDD, Test Driven Development) dando como resultado productos ALFAs. Cada dos Sprints el producto deja de ser ALFA y pasa a ser BETA, por lo que

estará listo para salir a un mercado de pruebas con el objetivo de que el equipo de desarrollo pueda retro-alimentarse rápidamente.

En el caso que un producto BETA alcance niveles altos de aceptación y tenga una aceptable calidad se convertirá en un producto estable y por tanto comenzará el proceso de registro.

Comercialización: El producto sale al mercado y comienza una interacción constante con los usuarios apareciendo criterios tanto negativos como positivos. Estos elementos a los que se les pueden añadir nuevas ideas salida de los usuarios, proporcionan funcionalidades mejoradas y en otros casos funcionalidades nuevas. Además obligan a un cambio que va asociado a la corrección de errores y a las adaptaciones requeridas a medida que va evolucionando el entorno del software. Los tipos de cambios pueden ser: Corrección, Adaptación, Mejora y Prevención.

2.1.5 Valores

Como metodología ágil posee los valores de la Alianza Ágil:

1. Individuos e interacciones más que procesos y herramientas.
2. Software operante más que documentaciones completas.
3. Colaboración con el cliente más que negociaciones contractuales.
4. Respuesta al cambio más que apegarse a una rigurosa planificación.

Otros valores de la metodología:

1. **Creatividad:** El elemento más importante que persigue SXP-J2ME es lograr un alto grado de creatividad en los integrantes del equipo para que así puedan surgir novedosas aplicaciones.
2. **Comunicación:** Desarrolla la comunicación entre sus integrantes partiendo de un respeto mutuo y una alta modestia.
3. **Humildad:** Todas las ideas son importantes, todos opinan.

2.1.6 Equipos

Los equipos son formados con un pequeño número de personas, entre 2 y 8 personas

donde la mayoría son estudiantes. Se distribuyen equitativamente, dentro de lo posible, a los que más experiencia tengan en el lenguaje de programación utilizado en los distintos equipos.

Como se ha mencionado anteriormente estos equipos se dividen en parejas. Cada pareja sentada en un ordenador implementa las clases diseñadas mediante las tarjetas CRC ayudándose uno con el otro e intercambiando ideas.

En ocasiones, cuando se está madurando una idea, método, algoritmo, etc, no es necesario estar frente de un ordenador, por lo que se recomienda reunirse cómodamente en un mesa disponible y debatir allí sobre el tema en cuestión. **(Ver Anexo J)**

2.1.7 Asignación de tareas

La asignación de tareas se realiza a través del Redmine, herramienta para la administración del proyecto. Cuando el proyecto es aprobado oficialmente una de las acciones a realizar de inmediato es la creación en el Redmine del proyecto aprobado. Una vez creado el proyecto he identificado el Visionario, Jefe de Producto y Jefe de Proyecto se asignan roles al equipo de desarrollo. Cada rol en el Redmine posee privilegios específicos.

2.1.8 Técnicas

Sprint de 15 días

Los Sprints cortos están bien. Permiten a la compañía ser “ágil”, es decir, cambiar de dirección frecuentemente. Sprints cortos = ciclo de feedback corto = más entregas y más frecuentes = más feedback del cliente = menos tiempo desarrollando en dirección incorrecta = aprender y mejorar más rápido, etc. [24]

En el desarrollo de aplicaciones para dispositivos móviles uno de los aspectos más importante es el time-to-maker por lo que se propone realizar Sprints de solo 15 días, logrando así una rápida salida al mercado de una versión funcional aunque no cuente con la totalidad de las funcionalidades concebidas.

“Descanso” entre Sprints

Además del propio descanso, hay otra buena razón para dejar algo de espacio entre Sprints. Después de la demo - versión ALFA en SXP-J2ME - y la retrospectiva tanto el Dueño de Producto – Visionario - como el Equipo estarán llenos de información y de ideas por digerir. Si se ponen de inmediato a planificar el próximo Sprint, es probable que nadie tenga la oportunidad de digerir ninguna información o lección aprendida, el Dueño de Producto no tendrá tiempo de ajustar sus prioridades después de la demo, etc. [24]

Se intenta introducir algo de descanso antes de empezar un nuevo Sprint (más específicamente, en el periodo después de terminar la retrospectiva del Sprint y antes de la siguiente reunión de planificación de Sprint). No siempre se consigue. [24]

Como mínimo, se intenta que la retrospectiva y la subsiguiente reunión de planificación de Sprint no ocurran el mismo día. Todo el mundo debería tener al menos una buena noche de sueño sin Sprint antes de comenzar el siguiente Sprint. [24]

Una forma de hacer esto son los “días de laboratorio” (o como se quiera llamar). Es decir, días en los que a los desarrolladores se les permite hacer esencialmente cualquier cosa que deseen (inspirado en Google). Por ejemplo, leer sobre las últimas herramientas y API’s, estudiar para una certificación, discutir asuntos técnicos con colegas, programar un proyecto personal como hobby, etc. [24]

El objetivo es tener un día de laboratorio entre cada Sprint. De esta forma obtenemos un descanso natural entre Sprints, y se tendrá un equipo de desarrollo que tendrá una oportunidad realista de mantener su conocimiento al día. Además, es un estupendo beneficio laboral. [24]

Diseño mediante tarjetas CRC

Las tarjetas CRC (Clase - Responsabilidad-Colaborador) **(Ver Anexo K)** son una herramienta con la cual a veces se asignan las responsabilidades y se indica una colaboración con otros objetos. Fueron inventadas por Kent Beck y Ward Cunningham, para que los diseñadores de objetos piensen en términos más abstractos sobre la asignación de responsabilidades y de las colaboraciones. Las tarjetas CRC son tarjetas de índices, una por cada clase, sobre las cuales se abrevian las responsabilidades de la clase y se anota una lista de los objetos con los que colaboran para desempeñarlas. Suelen elaborarse en una sesión de grupos pequeños donde los participantes

representan papeles de las diversas clases. Cada grupo tiene las tarjetas CRC correspondientes a las clases cuyo papel está desempeñando. La representación de papeles hace divertido aprender a pensar en función de objetos y de responsabilidades. [25]

Desarrollo orientado a pruebas (TDD, Test Driven Development)

Test-Driven Development es una técnica para aumentar la agilidad en el desarrollo del proyecto. La literatura empírica existente sobre TDD ha demostrado una mayor productividad y un código más robusto, entre otros beneficios importantes. [26]

El desarrollo ágil de software puede ser visto como un buen ajuste para el entorno móvil, con su alta volatilidad y fuertes necesidades de tiempo de salida al mercado. Las aplicaciones móviles son generalmente pequeñas y la mayoría de ellas son desarrollados por equipos de software pequeños. Las organizaciones que operan en este tipo de ambiente de negocios necesitan del desarrollo ágil para reaccionar rápidamente a las necesidades cambiantes del mercado. Los esfuerzos de las organizaciones que tratan de aumentar su capacidad de respuesta no serán suficientes si no se persigue la agilidad en todos los niveles de la organización, incluyendo el desarrollo asociado o de colaboración inter-institucional. Si las estructuras de organización no son compatibles con el intercambio de información rápida y los ciclos cortos de opinión, los beneficios de la agilidad no se consiguen. De hecho, varias organizaciones están muy interesados en la adopción de un conjunto de prácticas ágiles y los principios que estas utilizan. El desarrollo basado en pruebas (TDD) es una de las disímiles prácticas ágiles. [25]

El objetivo de TDD es ofrecer beneficios de agilidad a través de un conjunto de pruebas de unidad automatizadas.

Primeramente es implementada una prueba, luego se le añade el código para pasar esta prueba que se ha escrito. Cuando la prueba pasa, el código es refactorizado. La refactorización según Fowler es el proceso de realizar cambios al código sin cambiar su comportamiento externo, es decir, el código se ve alterado por los efectos de comentar, hacerlo más sencillo, o mejorando algún aspecto de calidad. Este ciclo se repite hasta que toda la funcionalidad está implementada. **(Ver Anexo E)**

Beck identifica algunos de los beneficios que se pueden obtener mediante la aplicación

de esta técnica de programación. A continuación se mencionan algunos:

1. Mantiene actualizada la documentación sobre el código.
2. Ayuda a los desarrolladores evitar el exceso de ingeniería mediante el establecimiento de un límite de lo que hay que implementar.
3. Crea una confianza al desarrollador de que el código escrito funciona correctamente.
4. Habilita la depuración rápida a través de un conjunto de pruebas que ayuda a identificar los defectos.

Control de las horas que faltan para terminar el objetivo del Sprint

Se debe tener un control de las horas que faltan para terminar el objetivo del Sprint. No es tan importante lo que se ha hecho como lo que falta por hacer. Además se hará una gráfica para determinar el progreso y comprobar qué tan finalista puede ser el equipo de desarrollo. **(Ver Anexo F)**

Importancia al factor Dedicación

En una universidad donde hay tareas que realizar no planificadas, es importante tener en cuenta el factor dedicación. Sería un error no considerar el tiempo que se deberá dedicar a tareas de otra índole y que se escapan del proyecto.

(DÍAS-HOMBRE DISPONIBLES) X (FACTOR DE DEDICACIÓN) = VELOCIDAD ESTIMADA

El factor de dedicación es una estimación de cómo de centrado va a estar el equipo. Un factor de dedicación bajo puede significar que el equipo espera encontrar muchas distracciones e impedimentos o que considera que sus propias estimaciones son optimistas. [24]

Es por ello que se propone incluir en el artefacto Minuta de Reunión, correspondiente a la segunda reunión de cada Sprint, una tabla que contenga una celda identificando al Sprint a que pertenece, los nombres de cada miembro del equipo de desarrollo de software así como los días disponibles, horas disponibles en cada día y total de horas disponibles correspondientes a cada uno de los integrantes.

Número de Sprint		*# del Sprint*	
Días laborables durante el Sprint		*# de días laborables durante el Sprint*	
Miembro del equipo	Días disponibles durante el Sprint	Horas disponibles por día	Total de horas disponibles
Nombre del miembro 1	#	#	#
Nombre del miembro 2	#	#	#
Nombre del miembro ...	#	#	#
Nombre del miembro n	#	#	#

Tabla 2: Control del factor Dedicación

2.1.9 Herramientas

SXP-J2ME está muy relacionada con las siguientes herramientas que a continuación se describen.

Bazaar

En el desarrollo de software es inevitable la edición de documentos o el cambio en algunos ficheros ya sean de configuración o de código, por lo que adquiere importancia utilizar una herramienta VCS (Sistema Controlador de Versiones).

Bazaar, es una de estas herramientas para el control de versiones y tiene la característica de soportar distintas formas de trabajo, en vez de obligar a que el equipo de desarrollo se adapte a una única forma posible. Las formas de trabajo se pueden cambiar, mezclar o ajustar como se necesite. Por ejemplo, un equipo puede decidir usar una forma tradicional "centralizada", pero complementarla con el modo en "pareja", es decir, dos desarrolladores intercambian directamente sus cambios cuando trabajan juntos.

Las formas de trabajo son muy independientes de la forma de almacenamiento. Si se quiere asegurar que todas las ramas se guarden en un repositorio central, hay formas de hacerlo, por ejemplo, llevando los cambios a un servidor central del que se hacen backups. En muchos equipos que utilizan herramientas de control de versiones, los desarrolladores suben el código al repositorio simplemente como backup o para etiquetarlo. Los sistemas de control de versiones distribuidos permiten hacer lo mismo, pero sin subir los cambios a la rama principal antes de lo debido, este es el caso de Bazaar, un software libre patrocinado por Canonical. [27]

Cuando se compara con SVN, que es una de las herramientas para el control de versiones de las más populares, podríamos decir que Bazaar presenta las siguientes ventajas:

1. Posee un mejor modelado de workflow's.
2. Mejores mezclas y posibilidades de hacer depender las mezclas de inteligencia humana.
3. Posibilidad de trabajar sin conexión, manteniendo el control de versiones local.
4. Posee logs más claros.

Redmine

Redmine es una aplicación web para la administración flexible de proyectos. Está escrito en Ruby. Además de que es multiplataforma soporta disímiles bases de datos. Es de código abierto siendo GNU Genral Public License v2 (GPL) su licencia. [28]

Posee características como: [29]

1. Soporta múltiple proyectos.
2. Control de acceso flexible basado en roles.
3. Posee notificaciones por correo electrónico.
4. Posibilita actualizar una wiki y foros de los proyectos.
5. Se integra con distintos controladores de versiones tales como SVN, CVS, Git, Mercurial, Bazaar and Darcs.
6. Soporta múltiples autentificaciones LDAP.
7. Soporta una gran variedad de lenguajes.
8. Soporta múltiples bases de datos como MySQL, PostgreSQL o SQLite.

2.1.10 Estándares

La mayoría de los programadores tienen su propio y distintivo estilo de programación. Pequeños detalles como la forma en la que tratan las excepciones, como comentan el código, cuando devuelven un valor null, etc. En algunos casos esta diferencia no importa, pero en otros puede conducir a una severa inconsistencia del diseño del

sistema y a un código difícil de leer. Un estándar de código ayuda a que esto no ocurra, siempre que se concentre en lo que realmente importa. [24]

2.2 Prácticas de SXP-J2ME

Las prácticas que intervienen en la metodología de desarrollo propuesta son las siguientes:

1. **Reuniones para controlar el Sprint terminado y planificar el siguiente:** Para el siguiente Sprint se realiza un juego de planeación teniendo en cuenta la estimación del esfuerzo.
2. **Release cortos:** entregas de versiones de aplicaciones que sean operativas, aunque no cuenten con todas las funcionalidades planificadas.
3. **Utilización de Metáforas:** Una metáfora es una historia compartida que describe cómo debería funcionar el sistema (conjunto de nombres que actúen o como vocabulario para hablar sobre el dominio del problema, ayudando a la nomenclatura de clases y métodos del sistema).
4. **Diseño simple:** Realizar diseños simples para evolucionar constantemente adicionando necesarias flexibilidades y eliminando innecesarias complejidades.
5. **Pruebas:** Los desarrolladores escriben pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión, el código escrito debe ejecutarse sin fallo para que el desarrollo continúe. El Visionario junto con el Encargado de Pruebas deben escribir pruebas que demuestren que las funcionalidades han sido desarrolladas correctamente.
6. **Participación activa de todos los miembros del proyecto:** Todos los miembros del proyecto pueden sugerir funcionalidades nuevas.
7. **Refactorización:** Los desarrolladores reestructuran el sistema eliminando código duplicado, simplificándolo o adicionándole flexibilidad.
8. **Programación en pares:** Se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. Se le otorga mayor importancia

a la calidad del código escrito de esta manera -el código es revisado y discutido mientras se escribe- que a la posible pérdida de productividad.

9. **Rápida retroalimentación (feedback):** Sacar versiones al mercado interno (versiones BETAs) para retroalimentarse rápidamente de los usuarios.
10. **Propiedad colectiva:** El código puede ser cambiado por cualquier integrante del equipo de desarrollo.
11. **Integración continua:** Se integra y se prueba todo el sistema varias veces en el día.
12. **40 horas semanales:** Trabajar en el proyecto 40 horas semanales y solo 40 horas, a razón de 8 horas diarias. No se trabajan horas extras en dos semanas seguidas. Si esto ocurre, probablemente está ocurriendo un problema que debe corregirse. El trabajo extra desmotiva al equipo.
13. **Cliente en la puerta:** Los clientes no están fuera ni dentro del proyecto. Si bien no se cuenta con un cliente específico en el equipo de desarrollo, se poseen varios usuarios bien cercas; los cuales pueden ser invitados y tomar el papel de cliente.
14. **Estándar de programación:** Utilización por parte del equipo de desarrollo de un único estándar de programación.
15. **Jefe de Proyecto (Scrum Master en Scrum):** Asegurar que el proyecto se está llevando a cabo de acuerdo con la metodología de desarrollo empleada y que todo funciona según lo planeado. Además, tener conocimiento del estado anímico de los integrantes del proyecto así como planificar actividades para incrementar el mismo es bienvenido.
16. **Equipo del Proyecto (Scrum Teams en Scrum):** Es el equipo del proyecto que tiene la autoridad para decidir cómo organizarse para cumplir con los objetivos de un Sprint.
17. **Lista de Reserva de Producto (similar a Product Backlog en Scrum):** es un documento que posee una lista priorizada que define el trabajo que se va a realizar en el proyecto. Esta lista puede crecer y modificarse a medida que se

obtiene más conocimiento acerca del producto (con la restricción de que solo puede cambiarse entre Sprint).

18. **Sprint:** pequeñas mejoras, unas tras otras. Se integra y se prueba todo el sistema varias veces en el día.

Conclusiones del capítulo

En el presente capítulo se ha descrito detalladamente las principales características de SXP-J2ME.

Luego de describir SXP-J2ME se puede concluir que se ha propuesto una metodología con tres fases y cuatro flujos de trabajo principales. Por sus características se considera ágil y además agrupa muchas de las llamadas “buenas prácticas de desarrollo de software” empleadas en las más importantes y utilizadas metodologías ágiles encontradas en la literatura.

Resulta interesante la práctica “Cliente en la puerta”, pues de cierto modo es un balance entre cliente fuera y cliente dentro del equipo de desarrollo; ya que en el desarrollo de este tipo de aplicaciones no existe, en la mayoría de los casos, un cliente bien definido. Aprovechando que los usuarios finales se encuentran cerca y dándole un alto nivel de importancia a la retroalimentación rápida, se considera a estos como clientes.

Capítulo III: Evaluación de SXP-J2ME

3.1 Métodos utilizados

Para evaluar SXP-J2ME se han empleado dos métodos, uno llamado 4-DAT (4 Dimensional Analytical Tool) propuesto por Asif Qumer y Brian Henderson-Sellers y otro propuesto por Rodríguez y Salmón. Este último, propuesto en la Universidad de las Ciencias Informáticas (UCI), tiene la característica de ser completamente cuantitativo.

4-DAT

4-DAT es basado en un estudio y evaluación de las diferentes definiciones contemporáneas. Sus autores, ofrecen la siguiente definición de agilidad de cualquier entidad: [30]

"La agilidad es el comportamiento persistente o la capacidad de una entidad sensible que muestra flexibilidad para adaptarse rápidamente a los cambios esperados o inesperados, consumiendo el menor tiempo posible, utilizando instrumentos económicos, sencillos y de calidad en un entorno dinámico."

Con el fin de beneficiarse de esta definición, dichos autores desarrollaron un marco de trabajo de cuatro dimensiones para cristalizar los principales atributos de agilidad: flexibilidad, velocidad, esbeltez, aprendizaje y capacidad de respuesta.

En consecuencia, mediante la aplicación de la definición anterior de agilidad a la noción de una metodología de desarrollo de software, se deriva la definición de un "método ágil", como: [30]

"Un método de desarrollo de software se dice que es un método ágil de desarrollo de software cuando se centra en las personas, es orientado a la comunicación, flexible (listo para su adaptación a la espera de un cambio inesperado en cualquier momento), rápida (estimula el rápido e iterativo desarrollo del producto en versiones pequeñas), delgado (se centra en acortar los plazos y costes y en la mejora de la calidad), sensible (reacciona adecuadamente a los cambios esperados e inesperados), y aprende (centrado en la mejora durante y después del desarrollo del producto)".

4-DAT facilita el examen de los métodos ágiles desde cuatro perspectivas o dimensiones: alcance de la metodología, caracterización de la agilidad, valores ágiles (Manifiesto Ágil) y caracterización del proceso de software. Aunque en la actualidad hay cuatro dimensiones evaluadas en el enfoque de 4-DAT, es ampliable en el hecho de que se pueden agregar o quitar dimensiones o elementos de las dimensiones, si se considera necesario en el futuro. [31]

La herramienta está diseñada para comparar y analizar los métodos ágiles. Un informe que se genera con la ayuda de 4-DAT se puede utilizar para la toma de decisiones con respecto a la adopción de un método ágil apropiado. [32]

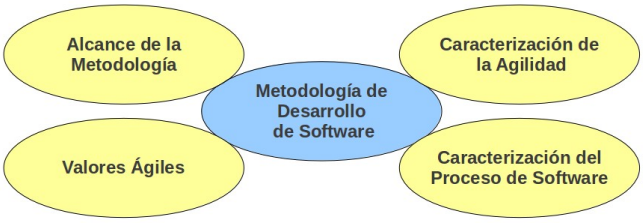

Ilustración 14: Visualización de las dimensiones de 4-DAT.

El nivel de significación de cada una de las dimensiones establecidas responde a la verificación de la metodología en el cumplimiento de la filosofía y la esencia de las Metodologías Ágiles en lo referente a las fases, las prácticas, los valores ágiles y la velocidad con que se debe desarrollar el producto de software. [33]

A continuación se describen las dimensiones propuestas en 4-DAT **(Ver Anexo G)**: [30]

Dimensión 1 Alcance de la metodología: Es un conjunto de elementos claves de alcance. Comprueba el apoyo de la metodología en términos de tamaño del proyecto, el tamaño del equipo, el estilo de desarrollo, el estilo de código, el entorno tecnológico, entorno físico, la cultura empresarial y el mecanismo de abstracción. Ayuda a comparar los métodos en un nivel alto.

Dimensión 2 Caracterización de la agilidad: Se proporciona un conjunto de características de agilidad que se derivan de la definición de agilidad descrita anteriormente. Se utiliza para comprobar la existencia de la agilidad en la metodología, tanto a nivel de proceso como de las prácticas. Esta es la única de las cuatro

dimensiones propuestas que es cuantitativa. En esta dimensión se construye una tabla en la que el valor de cada una de las celdas es de 0 o 1; estos se introducen en cada fase (para una evaluación de calidad) o, a un nivel más detallado, para cada práctica individual (también se le conoce como técnica). Las cinco características de agilidad son flexibilidad (AF), velocidad (SD), esbeltez (LS), aprendizaje (LG) y capacidad de respuesta (RS). Sobre la base de las evaluaciones cuantitativas (grado medio de agilidad) de las existente y bien conocidas metodologías ágiles como Scrum y XP, se puede ofrecer como una cifra aproximada un valor umbral de alrededor de 0.5 - 0.6 para que una metodología propuesta se pueda considerar como metodología ágil.

Dimensión 3 Valores Ágiles: Es un conjunto de seis valores ágiles, cuatro de ellos provistos del Manifiesto Ágil, un quinto valor proporcionado por [34] y el sexto valor de "mantener el proceso rentable" fue propuesto en [32], basado en el estudio de diferentes métodos ágiles. Esta tercera dimensión examina el apoyo de los valores ágiles en las diferentes prácticas de los métodos ágiles.

Dimensión 4 Caracterización del proceso de software: Es un conjunto de cuatro componentes que caracterizan el proceso de software. Hay dos componentes principales: el proceso de ingeniería de producto y el proceso de gestión de procesos. Un proceso de ingeniería de producto tiene un período de tres categorías: proceso de desarrollo, el proceso de gestión de proyectos y los procesos de apoyo [35]. En la Dimensión 4 se examinan las prácticas que apoyan estos cuatro procesos en los métodos ágiles.

Método cuantitativo

Por otra parte, el método propuesto por Rodríguez y Salmón, tiene la característica de ser netamente cuantitativo. Para su realización, sus autores se basaron en 4-DAT logrando llevar la totalidad de las dimensiones a forma cuantitativa; además de que se agregó una quinta dimensión llamada “Roles”.

Ambos métodos facilitan la obtención del grado de agilidad de los procedimientos metodológicos al que le sean aplicados y se pueden establecer consideraciones para tener en cuenta una determinada metodología a la hora de realizar un software, ya que son obtenidos valores concretos relacionados con la velocidad de desarrollo en las dimensiones definidas. [33]

3.2 Evaluación comparativa de XP, Scrum y SXP-J2ME mediante 4-DAT

Los datos correspondientes a las metodologías de desarrollo XP y Scrum que se utilizan en esta comparación han sido tomados del trabajo “COMPARATIVE EVALUATION OF XP AND SCRUM USING THE 4D ANALYTICAL TOOL (4-DAT)”, presentado por los creadores de 4-DAT (Asif Qumer y Brian Henderson-Sellers). [36]

3.2.1 Dimensión 1: Alcance de la metodología

Criterio	XP	Scrum	SXP-J2ME
Tamaño de los proyectos	Pequeños y medianos	Pequeños, medianos y grandes	Pequeños y medianos
Tamaño de equipo	Menor que 10	Múltiples equipos menores que 10	Múltiples equipos menores que 10
Estilo de desarrollo	Iterativo y rápido	Iterativo y rápido	Iterativo y rápido
Estilo de código	Limpio y sencillo	No especificado	Limpio y sencillo
Entorno tecnológico	Requiere rápida retroalimentación	No especificado	Requiere rápida retroalimentación
Entorno físico	Equipos en un mismo lugar y equipos distribuidos	No especificado	Equipos en un mismo lugar y equipos distribuidos
Cultura de negocio	Colaborativo y cooperativo	No especificado	Colaborativo y cooperativo
Mecanismos de abstracción	Orientado a objeto	Orientado a objeto	Orientado a objeto

Tabla 3: Alcance de las metodologías XP, Scrum y SXP-J2ME

3.2.2 Dimensión 2: Caracterización de la agilidad

El grado de agilidad (DA) depende de los términos flexibilidad (FY), velocidad (SD), esbeltez (LS), aprendizaje (LG) y capacidad de respuesta (RS).

Para calcular la agilidad en esta dimensión Qumer y Henderson-Sellers proponen la siguiente fórmula: [32]

$$DA\ (Object) = (1/m)\sum m\ DA(Object, Phase\ or\ Practices)$$

	Características de agilidad					
SXP-J2ME	FY	SD	LS	LG	RS	**Total**
(i) Fases						
Inicio	1	1	0	0	1	3
Elaboración-Construcción	1	1	0	1	1	4
Transición	1	1	0	1	1	4
Total	**3**	**3**	**0**	**2**	**3**	**11**
Grado de Agilidad	**3/3**	**3/3**	**0/3**	**2/3**	**3/3**	**11/ (3*5)**
(ii) Prácticas						
Reuniones para controlar el Sprint terminado y planificar el siguiente.	1	1	0	1	1	4
Release cortos	1	1	0	1	1	4
Utilización de Metáforas	0	1	1	0	0	2
Diseño simple	1	1	1	1	1	5
Pruebas	1	1	0	1	1	4
Participación activa de todos los miembros del proyecto	1	1	1	1	1	5
Refactorización	1	1	1	1	1	5
Programación en pares	1	0	0	1	1	3
Rápida retroalimentación	1	1	1	1	1	5
Propiedad colectiva	1	0	0	1	1	3
Integración continua	1	1	1	1	1	5
40 horas semanales	0	0	0	1	0	1
Cliente en la puerta	1	0	1	1	1	4
Estándar de programación	1	1	1	1	1	5
Jefe de Proyecto (Scrum Master en Scrum)	1	1	0	1	1	4
Equipo del Proyecto (Scrum Teams en Scrum)	1	1	0	1	1	4
Lista de Reserva de Producto (similar a Product Backlog en Scrum)	1	1	0	1	1	4
Sprint	1	1	0	1	1	4
Total	**16**	**14**	**8**	**17**	**16**	**71**
Grado de Agilidad	**16/18**	**14/18**	**8/18**	**17/18**	**16/18**	**71/ (18*5)**

Tabla 4: Grado de Agilidad de SXP-J2ME.

Teniendo en cuenta el grado de agilidad de XP y Scrum **(Ver Anexo H e I)** se obtienen los siguientes resultados.

	XP	Scrum	SXP-J2ME
Fases	0.70	0.60	0.73
Lugar	2	3	1
Prácticas	0.73	0.80	0.79
Lugar	3	1	2
Promedio (Fases y Prácticas)	0.72	0.70	0.76
Lugar	2	3	1

Tabla 5: Comparación del grado de agilidad entre XP, Scrum y SXP-J2ME.

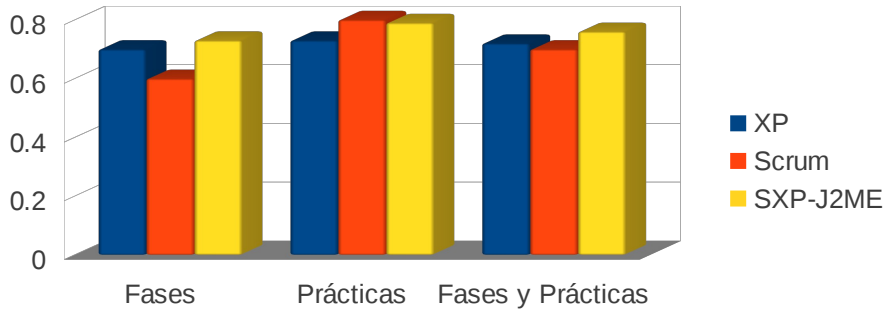

Ilustración 15: Gráfica representativa del grados de agilidad de XP, Scrum y SXP-J2ME.

3.2.3 Dimensión 3: Valores ágiles

La tercera dimensión de 4-DAT es sólo cualitativa y presentan los valores ágiles que son promulgadas por la metodología ágil bajo investigación. Se puede observar que tanto XP como Scrum ofrecen soporte para todos los valores iniciales ágiles, pero no son compatibles con los dos identificados recientemente por Qumer y Henderson Sellers. XP no ofrece ningún apoyo, ya sea para "mantener el proceso ágil" (lo que podría sugerir un eslabón perdido de la SPI (mejora de procesos software) de la comunidad) o para "mantener el coste del proceso efectivo" (un valor pragmático para la ágil adoptabilidad comercial). Esta última característica no se ve bien en Scrum. [36]

En el caso de SXP-J2ME, específicamente en el manteniendo del costo efectivo del proceso, se observa se le da gran importancia a la retroalimentación. Mediante ella se conoce si la aplicación tiene una aceptable aceptación por parte de los usuarios y en correspondencia una ganancia económica para la empresa desarrolladora de dicha aplicación. Otro elemento es el desarrollo de varios modelos en paralelo; contribuyendo

así a que llegué a los usuarios el producto que realmente quieran tener sin una demora considerable. Una de las características que ayuda a esta rapidez es la reutilización.

Valores Ágiles	XP	Scrum	SXP-J2ME
Individuos e iteraciones por encima de procesos y herramientas	-Juego de planificación. -Propiedad colectiva. -Cliente en el equipo de desarrollo. -Programación en pares.	-Scrum teams. -Sprint planning meeting. -Daily scrum meeting.	-Participación activa de todos los miembros del proyecto. -Propiedad colectiva. -Reunión para controlar el Sprint terminado y para planificar el siguiente. -Cliente en la puerta.
Software activo encima de documentación comprensiva.	-Release cortos. -Pruebas. -Integración continua.	-Sprint. -Revisión de Sprint.	-Sprint. -Entregas pequeñas mediante Sprints cortos . -Pruebas -Reunión para controlar el Sprint terminado y para planificar el siguiente. -Rápida retroalimentación . -Refactorización.
La colaboración con el cliente más que la negociación de un contrato.	-Juego de planificación. -Integración continua.	-Product backlog . -Sprint planning meeting.	-Desarrollo iterativo e incremental. -Participación activa de todos los miembros del proyecto .
Responder a los cambios más que seguir estrictamente un plan	-Metáforas. -Simple diseño. -Refactorización. -Estándar de codificación.	-Sprint review . -Sprint planning meeting.	-Utilización de Metáforas . -Simple diseño. -Refactorización. -Rápida retroalimentación. -Creación de varios modelos en paralelo. -Estándar de programación. -Reunión para controlar el Sprint terminado y planificar el siguiente .
Manteniendo procesos ágiles.		-Sprint review . -Daily scrum meeting.	-Reunión para controlar el Sprint terminado y planificar el siguiente . -Participación activa de todos los miembros del proyecto. -Rápida retroalimentación .
Manteniendo del costo efectivo del proceso.			-Rápida retroalimentación. -Creación de varios modelos en paralelo. -Reutilización constante.

Tabla 6: Grado de agilidad de XP, Scrum y SXP-J2ME en la dimensión 3.

3.2.4 Dimensión 4: Caracterización del proceso de software

En esta parte de la evaluación, se examinan las prácticas de XP, Scrum y SXP-J2ME para el apoyo a los procesos de software. En la tabla 6 se presenta un informe de evaluación para esta cuarta dimensión. Al igual que en la dimensión 3, la comparación es puramente cualitativa e informativa (descriptiva y no normativa).

Tanto XP como Scrum ofrecen prácticas para los procesos de desarrollo y de proyectos, pero no dicen nada sobre la gestión de configuración o de gestión de procesos. [36]

SXP-J2ME posee las mismas características, y no es de extrañar ya que es en parte una mezcla de ambas.

Proceso de software	XP	Scrum	SXP-J2ME
Proceso de desarrollo	-Release cortos. -Metáforas. -Diseño simple. -Pruebas. -Refactorización. -Programación en pares. -Propiedad colectiva. -Integración continua. -Cliente en el equipo de desarrollo. -Estándar de codificación.	-Scrum teams. -Product backlog. -Sprint. -Sprint review .	-Entregas pequeñas mediante Sprints cortos. -Utilización de Metáforas. -Diseño simple. -Pruebas. -Refactorización. -Programación en pares. -Propiedad colectiva. -Desarrollo iterativo e incremental . -Cliente en la puerta. -Estándar de codificación. -Rápida retroalimentación .
Proceso de gestión de proyecto	-Juego de planificación.	-Scrum Master. -Sprint planning meeting. -Daily scrum meeting.	-Participación activa de todos los miembros del proyecto . -Reunión para controlar el Sprint terminado y planificar el siguiente .
Proceso de la configuración del software	No especificado	No especificado	No especificado
Proceso de gestión de procesos	No especificado	No especificado	No especificado

Tabla 7: Proceso de software de XP, Scrum y SXP-J2ME. (Dimensión 4)

3.3 Evaluación de SXP-J2ME con un método cuantitativo basado en 4-DAT

Con la aplicación del método propuesto por Rodríguez y Salmón a los procedimientos ágiles, se logrará obtener que tan ágil son estos, en cuanto a los elementos principales que fueron definidos para el desarrollo de un software con la menor cantidad de artefactos producidos en cada fase, así como el cumplimiento que les dan las

Metodologías Ágiles a otros aspectos que son de vital importancia a la hora de realizar software con un límite de tiempo restringido y sobre todo que sea indispensable la entrega del producto funcional lo antes posible. [33]

La importancia del valor que pueda arrojar una determinada metodología ágil después de aplicarle el método, está substancialmente enfocado a la hora de la posible selección de una de estas para la realización de un determinado producto de software que requiera de un rápido desarrollo. [33]

3.3.1 Dimensión 1: Alcance de la metodología

En SXP-J2ME el proceso de captura de requisitos funcionales y no funcionales se logra a través de las Historias de Usuarios y la Ficha Técnica. Mientras que la definición de la arquitectura constituye unos de los objetivos esenciales dentro de la fase Inicio.

	Elementos	**Valor Asociado**
Análisis	Requisitos Funcionales	0.40
	Requisitos No Funcionales	0.20
	Diseño de Algoritmos	0
	Definición de la Arquitectura	0.30
Total		**0.90**

Tabla 8: Valor de SXP-J2ME en la Dimensión 1. Análisis.

SXP-J2ME cumple con el elemento establecido como básico en la etapa de diseño a través de las tarjetas CRC, por tanto obtiene el máximo valor asociado a la etapa del diseño y no está sujeta a ninguna penalización.

	Elementos	**Valor Asociado**
Diseño	Diagrama de Clases del Diseño	1
Total		**1**

Tabla 9: Valor de SXP-J2ME en la Dimensión 1. Diseño.

La fase Elaboración-Construcción se centra en la codificación, o sea, la implementación del diseño realizado a través de las tarjetas CRC; por tanto no recibirá ningún valor de los asignados para los tres primeros elementos que fueron seleccionados para esta etapa. El cuarto elemento se evalúa con 0.25 ya que SXP-

J2ME propone que en el artefacto Ficha Técnica quede plasmado el diagrama de despliegue.

Los objetivos de esta etapa al igual que XP lo complementa mediante la práctica Refactorización, actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios.

	Elementos	**Valor Asociado**
Implementación	Diagrama de Componentes	0
	Modelo de Implementación	0
	Subsistema de implementación	0
	Diagrama de Despliegue	0.25
Total		**0.25**

Tabla 10: Valor de SXP-J2ME en la Dimensión 1. Implementación.

La estrategia seguida por SXP-J2ME para cumplir como esencial en esta etapa está basada en la producción de código dirigido por pruebas. Las pruebas unitarias son establecidas antes de escribir el código y son ejecutadas constantemente ante cada modificación del sistema. En este contexto la automatización para apoyar esta actividad es de vital importancia. Además los integrantes del equipo de desarrollo que se desempeñen en el rol Encargado de Pruebas, junto con el Visionario, escriben y ejecutan las pruebas funcionales para cada Historia de Usuario con el objetivo de verificar que los requisitos establecidos en ellas se han cumplido satisfactoriamente .

	Elementos	**Valor Asociado**
Pruebas	Casos de Pruebas	0.50
	Plan de Pruebas	0.50
Total		**1**

Tabla 11: Valor de SXP-J2ME en la Dimensión 1. Pruebas.

Valor de la agilidad de SXP-J2ME en la dimensión Alcance de la metodología

VAD (Alcance de la metodología) = ∑ coeficiente de completitud de cada etapa – p.

p: penalización

VAD (Alcance de la metodología) = (0.90 + 1 + 0.25 + 1) – 0

VAD (Alcance de la metodología) = 3.15

3.3.2 Dimensión 2: Atributos para obtener la agilidad

El grado de agilidad de esta dimensión está basado en la cuantificación de los indicadores definidos para los parámetros que brindan la agilidad de un determinado procedimiento.

Parámetros Establecidos [33]

Velocidad (V): Elementos asociado con los procedimientos ágiles a la hora de realizar el software, como es el caso de la cantidad de roles, ya que las metodologías ágiles basan parte de su filosofía en este aspecto. Así como ser lo más subjetivo y eficiente a la hora de realizar las iteraciones en el tiempo definido.

Dinamismo (D): Es la velocidad con que se producen los cambios en los requerimientos. Los cambios de contexto, tanto sea de negocios, de la organización, técnicos, etc. Estos son considerados desde el punto de vista de cambios de requerimientos.

Aprendizaje (A): El indicador relacionado con este atributo se basa en el nivel de asimilación del conocimiento anterior actualizado y la experiencia de crear un ambiente de aprendizaje. El análisis de este indicador esta basado en la cooperación y aporte que pueden existir entre las distintas fases de los métodos ágiles.

Ligereza (L): Es la capacidad del equipo de desarrollo en producir lo más rápido posible y con la mejor calidad los aspectos relacionados con el lenguaje de modelado, o sea tener buena habilidad para obtener el modelo de lo que la metodología ágil en cuestión desee modelar. Esto en esencia se refiere a la capacidad y el adiestramiento de las personas encargadas en la tarea.

Flexibilidad (F): Se refiere a la capacidad de los procedimientos ágiles de enfrentarse a los cambios y ver a estos como algo común dentro del desarrollo de software y así dar la posibilidad al cliente de cambiar los requisitos cuando el desee. Este parámetro se puede sintetizar en uno de los principios básicos de las metodologías ágiles que no es más que responder al cambio antes de seguir estrictamente un plan, esto no quiere decir que no se planifique, pero si estar listos para enfrentar cualquier cambio en los requisitos, nuevas peticiones o cambios en general, para responder a ellos y rehacer

los planes ya que lo importante es satisfacer al cliente y cumplir con sus necesidades.

	V	D	A	L	F
Análisis		1	1	1	1
Diseño		0	0	1	1
Implementación		0	1	1	1
Pruebas		0	0.50	0	1
Total	1.50	1	2.50	3	4

Tabla 12: Valores de SXP-J2ME en la Dimensión 2.

VAD (Parámetros relacionados con la agilidad) = ∑ AT (Análisis, Diseño, Implementación y Pruebas) / **n** x **m**.

Al aplicarle la fórmula correspondiente a esta dimensión se obtiene que su agilidad es de **0.6**.

3.3.3 Dimensión 3: Valores Ágiles

Para cada uno de los valores ágiles se definieron elementos con el fin de analizar las prácticas establecidas por las Metodologías Ágiles existentes que garanticen a cada uno de estos valores. [33]

Ecuación para la obtención del grado de agilidad de esta Dimensión:

VAD (Valores Ágiles) = ∑ coeficiente de completitud de cada valor ágil - p.

El coeficiente de completitud no es más que el valor asociado a los elementos que se consideran necesarios en los valores ágiles.

	Elementos	Valor Asociado
Individuos e iteraciones por sobre los procesos y las herramientas	Mayor valor al equipo que al entorno	0.50
	Realización de iteraciones en todas las etapas	0.50
Total		**1**

Tabla 13: Valor del elemento esencial “Individuos e iteraciones por sobre los procesos y las herramientas” en SXP-J2ME.

	Elementos	Valor Asociado
Software activo encima de documentación comprensiva	Producción de documentos necesarios	0.20
	Entregas frecuentes	0.40
	Pruebas de funcionamiento al software	0.40
Total		**1**

Tabla 14: Valor del elemento esencial “Software activo encima de documentación comprensiva” en SXP-J2ME.

	Elementos	Valor Asociado
La colaboración con el cliente más que la negociación de un contrato	Frecuente relación con el cliente	1
Total		**1**

Tabla 15: Valor del elemento esencial “La colaboración con el cliente más que la negociación de un contrato” en SXP-J2ME.

	Elementos	Valor Asociado
Responder a los cambios más que seguir estrictamente un plan	Estrategia para responder a los cambios	0.75
	Planificación flexible	0
Total		**0.75**

Tabla 16: Valor del elemento esencial “Responder a los cambios más que seguir estrictamente un plan” en SXP-J2ME.

VAD (valores ágiles) = ∑ coeficiente de completitud de cada valor ágil - p.

VAD (valores ágiles) = (1 +1 + 1 + 0.75) = 3.75 – 0 = **3.75**

3.3.4 Dimensión 4: Caracterización del proceso de software

El proceso de desarrollo de software es un conjunto estructurado de actividades para desarrollar un sistema de software. Con el fin de garantizar el éxito requerido tanto por la parte del cliente como la del equipo de desarrolladores es de vital importancia contar con prácticas que garanticen que el transcurso del software se desarrolle bajo guías basadas en garantizar todos los aspectos concernientes con el ciclo de vida del producto, así como contar con un proceso de administración y de configuración que respalde las expectativas de lo que se desea lograr al final de terminado el software,

que no es más que un sistema funcional que responda a los requisitos establecidos por el cliente. Con el propósito de cuantificar las prácticas involucradas en el proceso del software fueron definidos los siguientes aspectos: Proceso de Desarrollo, Proceso de Administración y Control de Configuración. [33]

	Elementos	**Valor Asociado**
Proceso de Desarrollo	Prácticas para cubrir el ciclo de vida completo del software	1
Total		**1**

Tabla 17: Valor del elemento “Proceso de Desarrollo” en SXP-J2ME.

	Elementos	**Valor Asociado**
Proceso de Administración del proyecto	Planificación del proyecto	0.75
	Realización de reuniones frecuentemente	0.25
Total		**1**

Tabla 18: Valor del elemento “Proceso de Administración del proyecto” en SXP-J2ME.

	Elementos	**Valor Asociado**
Control de Configuración del Software	Administración de Configuración	0
Total		**0**

Tabla 19: Valor del elemento "Control de Configuración del Software" en SXP-J2ME.

VAD (caracterización del proceso de software) = ∑ coeficiente de completitud de coeficiente de completitud de aspecto que conforma el proceso de software - p
VAD (caracterización del proceso de software) = (1 + 1 + 0) =2 – 0 = **2**

3.3.5 Dimensión 5: Roles

Para el desarrollo de software con Metodologías Ágiles, se deben tener en cuenta la cantidad y definición de los roles como parte de la esencia de la filosofía de las metodologías livianas. La evaluación cuantitativa de esta dimensión está basada en si los roles definidos por la metodología a evaluar se corresponden con los establecidos como básicos para cualquier proyecto. La fundamentación de la definición de los roles seleccionados como los esenciales estuvo basado en los resultados objetivos que se obtuvieron. El establecimiento de estos roles como esenciales también estuvo sujeto a

la valoración de las experiencias con que deben contar cada uno de los miembros del equipo que se disponga a desarrollar un determinado software bajo estas metodologías. [33]

El fundamento de la selección de los roles básicos (Líder del proyecto, Analista, Diseñador, Programador, Arquitecto y Probador) para garantizar agilidad, partió de la premisa de que con ellos se puede realizar un software de forma integra, siempre y cuando cuenten con los conocimientos necesarios. Por la importancia de cada uno se le asignó de forma equitativa el mismo valor de rango. [33]

Roles Básicos	Valor Asociado
Líder del Proyecto	0.16
Analista	0.16
Diseñador	0.16
Programador	0.16
Arquitecto	0.16
Probador	0.16
Penalización	0.40
Total	**0.60**

En el desarrollo del método se decidió por parte de los autores aplicarle una penalización a las metodologías de desarrollo de software que establecieran más roles de los considerados como básicos. Como la metodología propuesta define 4 roles que no están relacionados con los establecidos, debe ser penalizada con 0.40 o sea 0.10 por cada rol de más que define.

3.3.6 Agilidad Total

GAM (XP) = 1/5 ∑ VAD (Dimensión i)

GAM (XP) = 1/5 * (3.15 + 0.6 + 3.75 + 2 + 0.60)

GAM (XP) = 2.02

3.3.7 Comparación con RUP, XP y Scrum

Rodríguez y Salmón en el trabajo que presentan [33] realizan el cálculo correspondiente para determinar la agilidad de algunas metodologías entre las que se encuentran RUP, XP y Scrum; arrojando los siguientes resultados:

- RUP: 1.11

- XP: 1.89
- Scrum: 1.58

Si se comparan estos datos con los obtenidos en SXP-J2ME se observan resultados positivos.

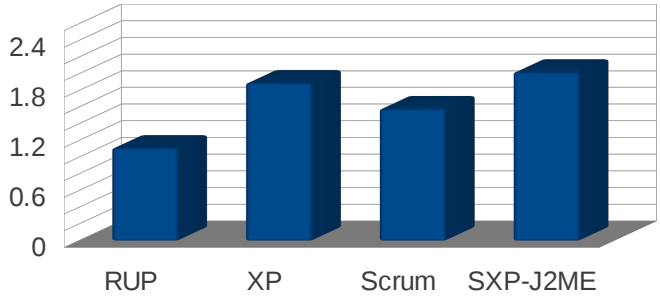

Ilustración 16: Comparación de los resultados obtenidos de las metodología RUP, XP, Scrum y SXP-J2ME al aplicarle el método de evaluación cuantitativo propuesto por Rodríguez y Salmón.

Conclusiones del capítulo

En el presente capítulo de han descrito dos métodos evaluadores de metodologías ágiles y mediante ellos se ha evaluado a SXP-J2ME. Luego se han comparado los resultados obtenidos con los de las metodologías XP y Scrum encontrados en la literatura, obteniendo resultados levemente superiores.

Conclusiones

Las metodologías de desarrollo de software ágiles permiten a los pequeños grupos de desarrollo de software concentrarse en la tarea de construir aplicaciones fomentando prácticas de fácil adopción y un entorno ordenado que ayude a que las personas trabajen mejor y permita que los proyectos finalicen exitosamente. SXP-J2ME, metodología propuesta en esta tesis, avanza en el conocimiento teórico de estos procesos reuniendo y proponiendo prácticas que contribuyan a la implementación y posterior adaptación del proceso a la realidad de la línea de productos “Aplicaciones J2ME para la Cultura y el Patrimonio”.

En el presente trabajo, se describió de forma breve el surgimiento de las metodologías ágiles, además de caracterizaron las más importantes y utilizadas como XP, Scrum, Cristal Clear, FDD y SXP. Gracias a este capítulo se logró ubicar al lector dentro del universo de las metodologías ágiles de desarrollo de software.

Posteriormente, en el capítulo dos, se describió la metodología ágil de desarrollo de software SXP-J2ME teniendo en cuanta los elementos descriptivos sugeridos por Alistair Cockburn. Con esta descripción se pretende lograr un elevado entendimiento por parte del lector del funcionamiento de SXP-J2ME; haciendo énfasis en las buenas prácticas recolectadas en la misma. Además se trató de aprender de los viejos errores y de generar un repositorio de buenas prácticas.

Finalmente en el capítulo tres se presentaron dos métodos para evaluar una determinada metodología de desarrollo de software, así como los resultados obtenidos al evaluar SXP-J2ME mediante ambos métodos. También se compararon dichos resultados con algunos ya obtenido al evaluar las metodologías de desarrollo de software más utilizadas como RUP, XP y Scrum. Estas evaluaciones resultaron positivas teniendo en cuanta que SXP-J2ME posee rasgos que la hacen más adaptable a la línea de productos “Aplicaciones J2ME para la Cultura y el Patrimonio” perteneciente a la Facultad Regional Granma y los valores obtenidos al evaluarla resultaron levemente superiores a XP y Scrum.

Recomendaciones

Se recomienda aplicar SXP-J2ME en los diferentes proyectos que conforman la línea de productos “Aplicaciones J2ME para la Cultura y el Patrimonio” perteneciente a la Facultad Regional Granma y luego realizar una comparación de los resultados alcanzados haciendo uso de SXP con los obtenidos con SXP-J2ME.

Además se sugiere evaluar la posibilidad de adicionar nuevas prácticas de desarrollo de software que ayuden en la formación profesional de los estudiantes y equipo de desarrollo en general que empleen SXP-J2ME y seleccionar otras que puedan contribuir a una retroalimentación más efectiva entre los usuarios y el equipo de desarrollo.

También se exhorta a la elaboración de un expediente de proyecto que recoja las plantillas de los artefactos propuestos en el presente trabajo.

Referencia Bibliográfica

1. PEÑALVER, G, MENESES, A and GARCÍA, S. SXP, Metodología Ágil para el Desarrollo de Software. In: 1er Congreso Iberoamericano de Ingeniería de Proyectos. Chile: s.n., 2010.

2. Inauguradas tres facultades de la Universidad de Ciencias Informáticas - Cuba - Juventud Rebelde - Diario de la juventud cubana. In: [online]. [Accessed 3 March 2012]. Available from: http://www.juventudrebelde.cu/cuba/2007-04-05/inauguradas-tres-facultades-de-la-universidad-de-ciencias-informaticas-/.

3. CARRILES, O. Desarrollo de aplicaciones JAVA para dispositivos móviles. S.l.: s.n.

4. BLANCO, P, CAMARERO, J, FUMERO, A, WERTERSKI, A and RODRÍGUEZ, P. Metodología de desarrollo ágil para sistemas móviles. Introducción al desarrollo con Android y el iPhone. In: Universidad Politécnica de Madrid: s.n., 2009.

5. Android Overview | Open Handset Alliance. In: [online]. [Accessed 6 2012]. Available from: http://www.openhandsetalliance.com/android_overview.html.

6. Android Steals Market Share from iPhone. In: [online]. [Accessed 6 2012]. Available from: http://www.readwriteweb.com/archives/android_steals_market_share_from_iphone.php.

7. GARZÓN, JP and GONZALEZ, E. BESA/ME: Plataforma para desarrollo de aplicaciones multi-agente sobre dispositivos móviles con JME. In: Revista Avances en Sistemas e Informática. 2009, Vol. 6, no. 3, pp. 12.

8. DE JODE, M, ALLIN, J, HOLLAND, D, NEWMAN, A and TURFUS, C. Programming Java 2 Micro Edition on Symbian OS. A developer's guide to MIDP 2.0. S.l.: John Wiley & Sons Ltd, 2004.

9. SELECTING A DEVELOPMENT APPROACH. February 2008. S.l.: s.n.

10. BOEHM, B and TURNER, R. Balancing Agility and Discipline: A Guide for the Perplexed. S.l.: Addison Wesley, 2003. ISBN 0-321-18612-5.

11. KRUCHTEN, P. What Is the Rational Unified Process? 2001. S.l.: s.n.

12. CALDERÓN, A and DÁMARIS, S. Metodologías Ágiles. 2007. S.l.: s.n.

13. Principios del Manifiesto Ágil. In: [online]. [Accessed 13 February 2012]. Available from: http://www.agilemanifesto.org/iso/es/principles.html.

14. BECK, K. Extreme Programming Explained. S.l.: s.n., 1999. ISBN 0201616416.

15. SCHENONE, MH. Diseño de una Metodología Ágil de Desarrollo de Software.

Tesis de Grado. S.l.: Universidad de Buenos Aires, 2004.

16. COCKBURN, A. Metodologías Crystal (Crystal Methodologies) I « Jummp. In: [online]. [Accessed 13 February 2012]. Available from: http://jummp.wordpress.com/2011/05/31/alistair-cockburn-metodologias-crystal-crystal-methodologies-i/.

17. ABRAHAMSSON, P, SALO, O, RONKAINEN, J and WARSTA, J. Agile software development methods. In: S.l.: s.n., 2002.

18. CANÓS, P, LETELIER, P and PENADÉS, MC. Métodologías Ágiles en el Desarrollo de Software [online]. S.l.: s.n. Available from: www.willydev.net/descargas/prev/TodoAgil.Pdf.

19. RAHIMIAN, V and RAMSIN, R. Designing and agile methodology for mobile software development: a hybrid method engineering approach. In: Second International Conference on Research Challenges in Information Science. Marrakech: s.n., 2008. pp. 337–342.

20. ALFONSO, AP, REGATEIRO, FS and SILVA, MJ. Dynamic Channels: a New Development Methodology for Mobile Computing Applications. 1998. S.l.: s.n. Universidade de Lisboa

21. SPATARU, AC. Agile Development Methods for Mobile Applications. S.l.: University of Edinburgh, 2010.

22. COCKBURN, A. Surviving object oriented projects. In: Addison-Wesley Object Technology Series. 1998,

23. COCKBURN, A. Agile Software Development. S.l.: s.n., 2000. Highsmith Series Editors.

24. KNIBERG, H. Scrum y XP desde las trincheras. S.l.: s.n., [no date].

25. LARMAN, C. UML y Patrones: Introducción al análisis y diseño orientado a objetos. S.l.: Prentice Hall, 1999. ISBN 0-13-748880-7.

26. ABRAHAMSSON, P, HANHINEVA, A and JAALINOJA, J. X Improving Business Agility Through Technical Solutions: A Case Study on Test-Driven Development in Mobile Software Development. 2005. S.l.: s.n.

27. Bazaar. In: [online]. [Accessed 6 April 2012]. Available from: http://bazaar.canonical.com/en/.

28. Overview - Redmine. In: [online]. [Accessed 6 April 2012]. Available from: http://www.redmine.org/.

29. Features - Redmine. In: [online]. [Accessed 6 April 2012]. Available from: http://www.redmine.org/projects/redmine/wiki/Features.

30. QUMER, A and HENDERSON-SELLERS, B. An evaluation of the degree of agility in six agile methods and its applicability for method engineering. In: Elsevier Inc. 2008, pp. 280–299.

31. QUMER, A and HENDERSON-SELLERS, B. A framework to support the evaluation, adoption and improvement of agile methods in practice. In: Elsevier Inc. [online]. 2008, pp. 1899–1919. DOI 10.1016/j.jss.2007.12.806. Available from: www.sciencedirect.com.

32. QUMER, A and HENDERSON-SELLERS, B. Measuring Agility and Adoptability of Agile Methods: A 4-Dimensional Analytical Tool. In: IADIS International Conference Applied Computing. S.l.: s.n., 2006. pp. 503–507.

33. RODRIGUEZ, J and SALMON, R. Método Cuantitativo de Evaluación de Metodologías Ágiles. Tesis de Grado. La Habana: Universidad de las Ciencias Informáticas (UCI), 2008.

34. KOCH, AS. Agile Software Develoment: Evaluating the Methods for Your Organization. London: Artech House, Inc., 2005.

35. JALOTE, P. An Integrated Approach to Software Engineering. New York: SpringerVerlag, 1997.

36. QUMER, A and HENDERSON-SELLERS, B. Comparative Evaluation of XP and SCRUM using the 4D Analytical Tool (4-DAT). In: European and Mediterranean Conference on Information Systems (EMCIS). Alicante, Spain: s.n., 2006.

Anexos

Anexo A

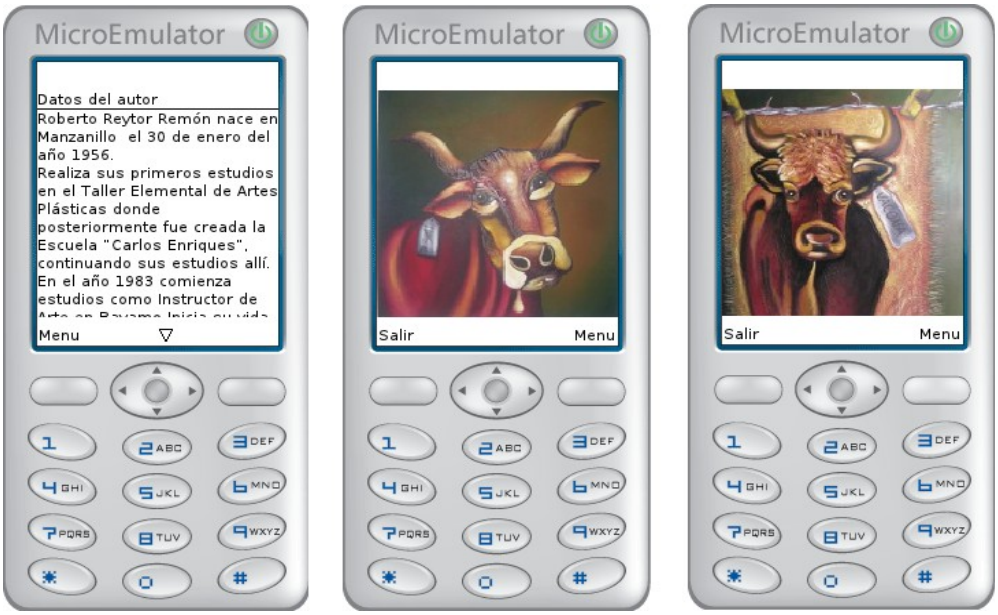

Ilustración 17: Aplicación "Galería Virtual Vacas de Roberto Reytor"

Anexo B

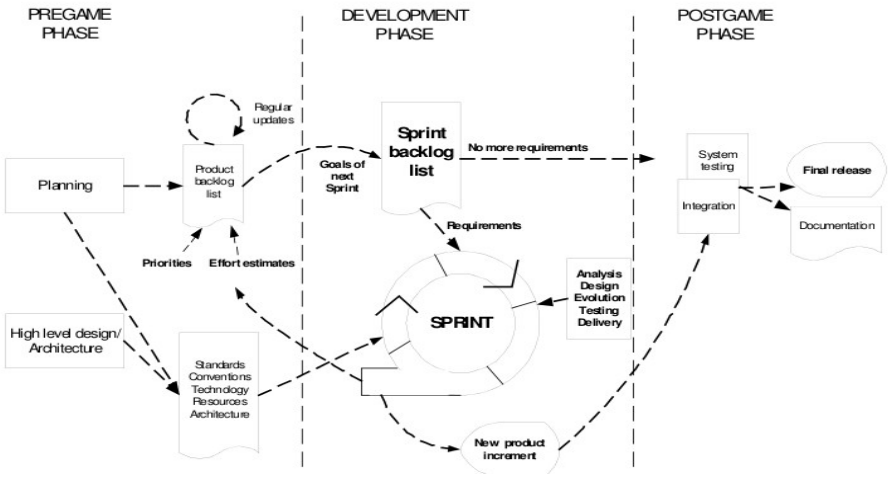

Ilustración 18: Proceso Scrum (Abrahamsson, 2002)

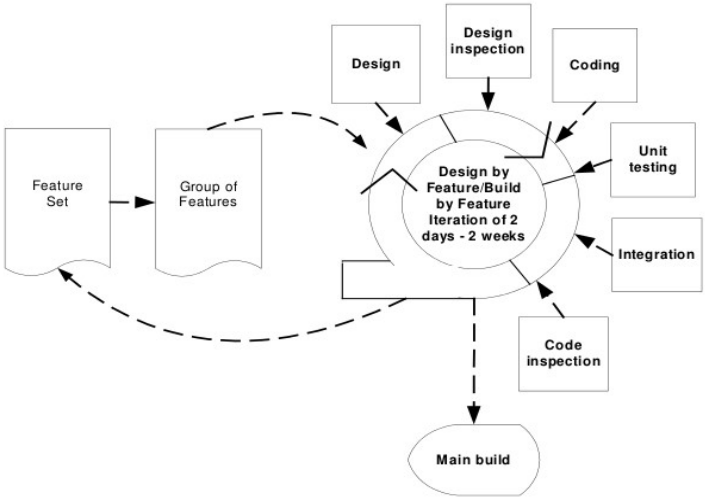

Ilustración 19: Diseño y construcción por funcionalidad de FDD (Abrahamsson, 2002)

Anexo C

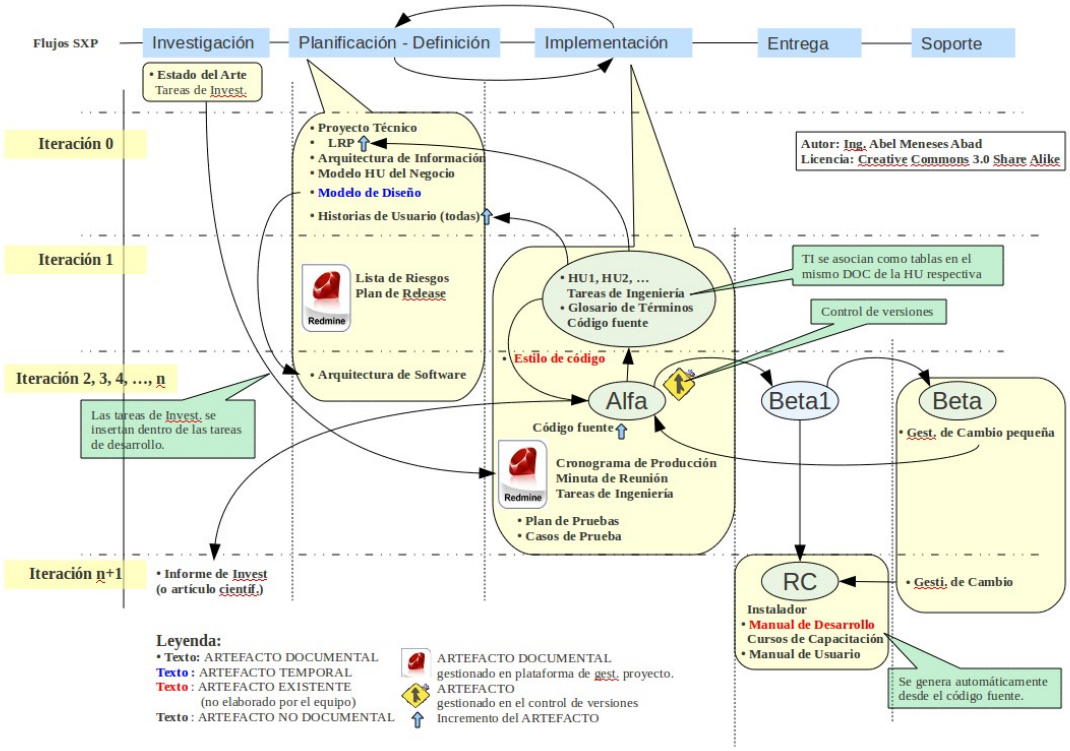

Ilustración 20: Guión de la metodología SXP (Meneses, 2011)

Anexo D

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código.	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.
Especialmente preparados para cambios durante el proyecto.	Cierta resistencia a los cambios.
Impuestas internamente (por el equipo).	Impuestas externamente.
Proceso menos controlado, con pocos principios.	Proceso mucho más controlado, con numerosas políticas/normas.
No existe contrato tradicional o al menos es bastante flexible.	Existe un contrato prefijado.
El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones.
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio.	Grupos grandes y posiblemente distribuidos.
Pocos artefactos.	Más artefactos.
Pocos roles.	Más roles.
Menos énfasis en la arquitectura del software.	La arquitectura del software es esencial y se expresa mediante modelos.

Tabla 20: Diferencias entre metodologías ágiles y no ágiles.

Factor	Metodologías Ágiles	Metodologías Tradicionales
Tamaño	Bien adaptado a los productos y equipos pequeños.	Métodos evolucionado para la manipulación de productos y equipos grandes.
Criticalidad	Probado en los productos de seguridad crítica. Posee dificultades potenciales como un diseño simple y poca documentación.	Métodos evolucionado para la manipulación de productos altamente críticos.
Dinamismo	El diseño simple y refactorización continua son excelentes para entornos altamente dinámicos, pero una fuente de re-trabajo puede ser caro para los entornos de gran estabilidad.	Los planes detallados y Diseño de Big Up Front excelente para el medio muy estable, pero una fuente de trabajo costoso para los entornos altamente dinámicos.
Personal	Requiere la presencia continua de clientes.	Necesidad de clientes en el equipo de desarrollo durante la definición del proyecto; pero puede trabajar con menos más adelante en el proyecto, a menos que el entorno sea muy dinámico.
Cultura	Prospera en una cultura donde la gente se sienta cómoda y fortalecida por tener muchos grados de libertad. (Prosperando en el caos)	Prospera en una cultura donde la gente se sienta cómoda y fortalecida por tener sus funciones bien definidas por las políticas y procedimientos. (Prosperar en el orden)

Tabla 21: Comportamiento según los factores de las metodología ágiles y las tradicionales.

Anexo E

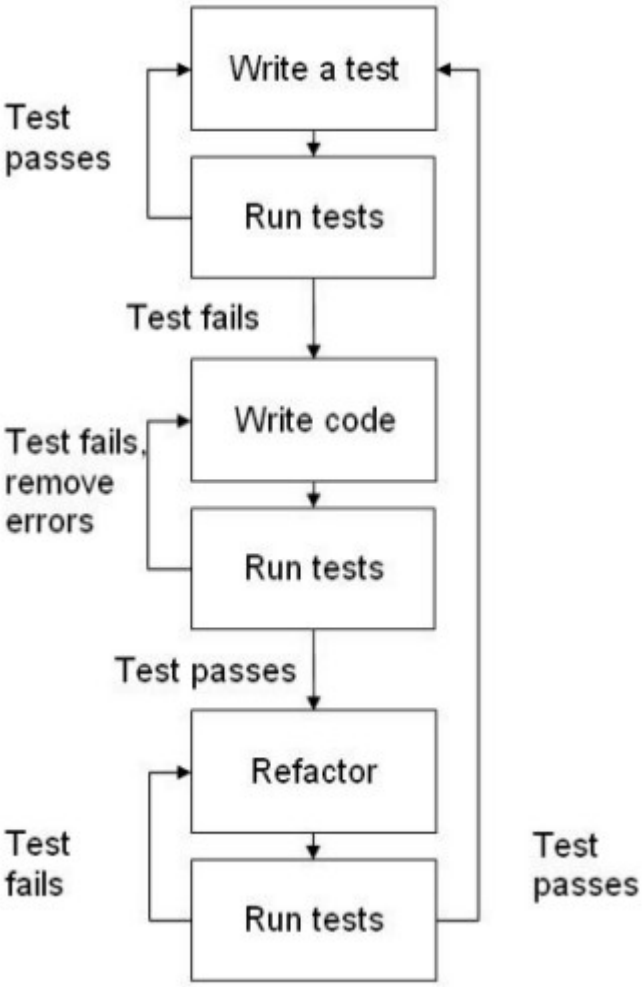

Ilustración 21: Pasos de TDD (Abrahanssom, 2005)

Anexo F

				Nuevo esfuerzo estimado Lo que queda al final del día...					
Elemento de la Pila de Producto	**Tarea del Sprint**	**Voluntario**	**Esfuerzo estimado inicial**	**1**	**2**	**3**	**4**	**5**	**6**
Como comprador quiero poner un libro en el carrito de la compra	modificar base de datos	Sanjay	5	4	3	0	0	0	
	crear página web (interfaz de usuario)	Jing	3	3	3	2	0	0	
	crear página web (lógica Javascript)	Tracy & Sam	2	2	2	2	1	0	
	escribir pruebas de aceptación automáticas	Sarah	5	5	5	5	5	0	
	actualizar la página de ayuda del comprador	Sanjay & Jing	3	3	3	3	3	0	
	. . .								
Mejorar el rendimiento de procesamiento de transacciones	juntar el código DCP y completar los test del nivel de capa		5	5	5	5	5	5	
	completar la orden de máquina para pRank		3	3	8	8	8	8	
	Cambiar el DCP y el lector para usar el API http de pRank		5	5	5	5	5	5	
			...						
		Total (personas-hora)		50	49	48	44	43	34

Ilustración 22: Ejemplo de control de horas restantes para culminar los objetivos del un Sprint

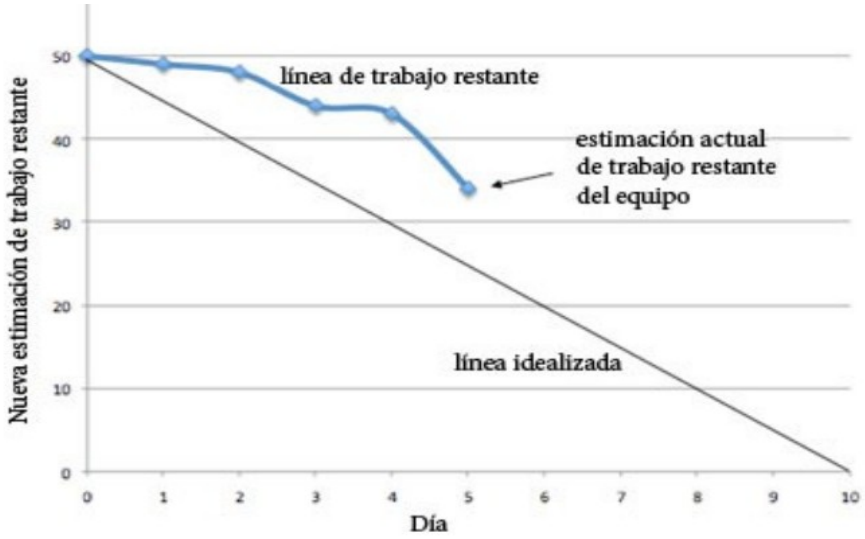

Ilustración 23: Ejemplo de gráfica para visualizar las horas restantes para el cumplimientos de los objetivos de un Sprint.

Anexo G

Scope	Description
1. Project Size	Does the method specify support for small, medium or large projects (business or other)?
2. Team Size	Does the method support for small or large teams (single or multiple teams)?
3. Development Style	Which development style (iterative, rapid) does the method cover?
4. Code Style	Does the method specify code style (simple or complex)?
5. Technology Environment	Which technology environment (tools, compilers) does the method specify?
6. Physical Environment	Which physical environment (co-located or distributed) does the method specify?
7. Business Culture	What type of business culture (collaborative, cooperative or non-collaborative) does the method specifiy?
8. Abstraction Mechanism	Does the method specify abstraction mechanism (object-oriented, agent-oriented)?

Ilustración 24: Dimensión 1 de 4-DAT. Alcance de la metodología. (Qumer, 2006)

Features	Description
1. Flexibility	Does the method accommodate expected or unexpected changes?
2. Speed	Does the method produce results quickly?
3. Leanness	Does the method follow shortest time span, use economical, simple and quality instruments for production?
4. Learning	Does the method apply updated prior knowledge and experience to learn?
5. Responsiveness	Does the method exhibit sensitiveness?

Ilustración 25: Dimensión 2 de 4-DAT. Caracterización de la Agilidad. (Qumer, 2006)

Agile values	Description
1. Individuals and interactions over processes and tools	Which practices value people and interaction over processes and tools?
2. Working software over comprehensive documentation	Which practices value working software over comprehensive documentation?
3. Customer collaboration over contract negotiation	Which practices value customer collaboration over contract negotiation?
4. Responding to change over following a plan	Which practices value responding to change over following a plan?
5. Keeping the process agile	Which practices helps in keeping the process agile?
6. Keeping the process cost effective	Which practices helps in keeping the process cost effective?

Ilustración 26: Dimensión 3 de 4-DAT. Caracterización de los Valores Ágiles. (Qumer, 2006)

Process	Description
1. Development Process	Which practices cover the main life cycle process and testing (Quality Assurance)?
2. Project Management Process	Which practices cover the overall management of the project?
3. Software Configuration Control Process / Support Process	Which practices cover the process that enables configuration management?
4. Process Management Process	Which practices cover the process that is required to manage the process itself?

Ilustración 27: Dimensión 4 de 4-DAT. Caracterización del Proceso de Software. (Qumer, 2006)

Anexo H

	Agility Features					
XP	FY	SD	LS	LG	RS	**Total**
(i) Phases						
Exploration	1	1	0	1	1	4
Planning	1	1	0	1	1	4
Iteration to release	1	1	0	1	1	4
Productionizing	1	1	1	1	1	5
Maintenance	1	0	0	1	1	3
Death	0	1	0	0	0	1
Total	**5**	**5**	**1**	**5**	**5**	**21**
Degree of Agility	**5/6**	**5/6**	**1/6**	**5/6**	**5/6**	**21/(6*5)**
(ii) Practices						
The Planning Game	1	1	0	1	1	4
Short Release	1	1		1	1	4
Metaphor	0	1	1	0	0	2
Simple Design	1	1	1	1	1	5
Testing	1	1	0	1	1	4
Refactoring	1	1	1	1	1	5
Pair Programming	1	0	0	1	1	3
Collective Ownership	1	0	0	1	1	3
Continuous Integration	1	1	1	1	1	5
40-Hour Week	0	0	0	1	0	1
On-site Customer	1	0	0	1	1	3
Coding Standards	1	1	1	1	1	5
Total	**10**	**8**	**5**	**11**	**10**	**44**
Degree of Agility	**10/12**	**8/12**	**5/12**	**11/12**	**10/12**	**44/(12*5)**

Tabla 22: Grado de agilidad de XP. Dimensión 2. (Qumer, 2006)

Anexo I

	Agility Features					
Scrum	FY	SD	LS	LG	RS	**Total**
(i) Phases						
Pre-Game	1	1	0	1	1	4
Development	1	1	0	1	1	4
Post-Game	0	1	0	0	0	1
Total	**2**	**3**	**0**	**2**	**2**	**9**
Degree of Agility	**2/3**	**3/3**	**0/3**	**2/3**	**2/3**	**9/(3*5)**
(ii) Practices						
Scrum Master	1	1	0	1	1	4
Scrum Teams	1	1	0	1	1	4
Product Backlog	1	1	0	1	1	4
Sprint	1	1	0	1	1	4
Sprint Planning Meeting	1	1	0	1	1	4
Daily Scrum Meeting	1	1	0	1	1	4
Sprint Review	1	1	0	1	1	4
Total	**7**	**7**	**0**	**7**	**7**	**28**
Degree of Agility	**7/7**	**7/7**	**0/7**	**7/7**	**7/7**	**28/(7*5)**

Tabla 23: Grado de agilidad de Scrum. Dimensión 2. (Qumer, 2006)

Anexo J

Ilustración 28: Local perteneciente al Centro de Desarrollo de Software (UCI) en la provincia Villa Clara.

Ilustración 29: Mesa para reuniones, intercambio de ideas, etc, en el laboratorio perteneciente al Centro de Desarrollo de Software (UCI) en la provincia Villa Clara.

Anexo K

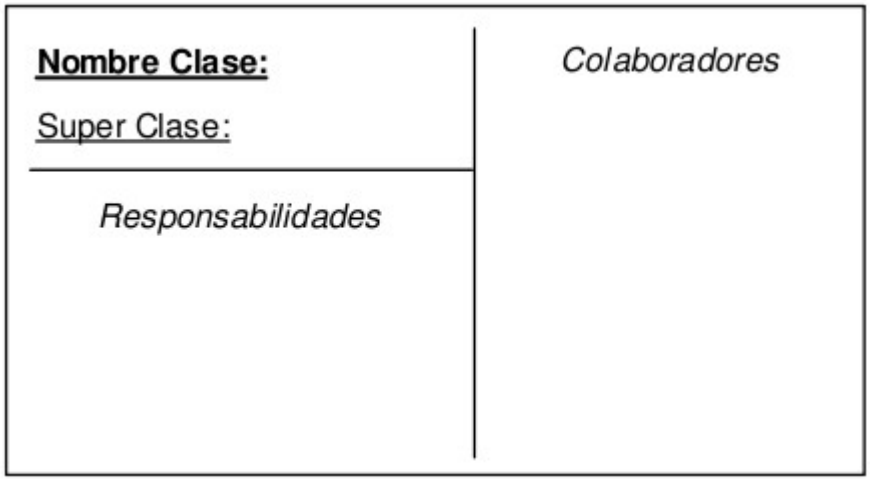

Ilustración 30: Ejemplo de Tarjeta CRC