Universidad Tecnológica Nacional
Facultad Regional Buenos Aires

Tesis de Maestría en
Ingeniería en Sistemas de Información

CAPA-3: Una innovadora metodología para el desarrollo de software en ambientes de trabajo virtuales

Autor: **MITARITONNA, Alejandro Daniel**

Director: **LÓPEZ, Carlos Alberto**

Co-Director: **BOIAROV, Sonia**

Ciudad Autónoma de Buenos Aires, 2010

Dedicatoria

*A la memoria de mi padre **Héctor Víctor** y a la de mis abuelos **Florencio**, **Carmen y Nicolás**, que Dios me los cuide por siempre.*

*A mi esposa **Adriana**, a mi hijo **Germán**, a mi madre **Hilda** y a mi hermana **María Teresa**; por su amor incondicional y su paciencia, pero sobre todo por creer en mí y apoyarme en la lucha por mis propios sueños.*

Agradecimiento

Un inmenso agradecimiento a mi director de tesis, Mag. Carlos López y a mi co-directora de tesis, Mag. Sonia Boiarov, por su paciencia, apoyo y ayuda durante la realización de este proyecto.

A todo el personal de la empresa donde se realizó este proyecto; por su disposición y colaboración, haciendo posible la culminación de este trabajo.

A mis colegas y compañeros de curso, por los momentos tan especiales durante aquellos años: Oscar Bruno, Vicente Santangelo, María Eugenia López Elguea y Lorena Arias Esteban.

Finalmente a toda mi familia; gracias por ayudarme en los momentos donde necesité una mano amiga y por apoyarme en la lucha por mis metas: Adriana, Germán, Marité, Hilda y a todos los que olvide mencionar, gracias de todo corazón.

Índice de Contenidos

Lista de Tablas

Lista de Gráficos

Resumen

Dada la importancia de la externalización del desarrollo de software, al convertirse como uno de los factores de éxito críticos de crecimiento y de lo beneficioso para cualquier organización, hay poca investigación hecha con respecto a que clase de metodología adoptar en los procesos de desarrollo de software en ambiente virtuales y dispersos geográficamente. Los análisis realizados hasta la fecha sirven como guías básicas de cómo se deberían realizar las tareas de desarrollo de software pero no entran en detalle en la gestión, en la metodología de desarrollo de software de equipos dispersos geográficamente, el uso de herramientas y menos aún en el factor humano que conforma los equipos de desarrollo de software distribuido y que son clave para éxito de los proyectos.

El foco de esta investigación se centra en diseñar un eje regulador entre las tareas de control y gestión de los proyectos, las metodologías de desarrollo de software adecuadas y por último los tipos de herramientas que darán soporte a toda la metodología descripta en ambientes distribuidos. Por lo tanto se busca la relación en tres capas entre gestión, metodología y herramientas como un proceso completo cuyo objetivo es simplificar todas las tareas que forman parte de proceso de desarrollo de software y en donde los recursos humanos juegan un papel relevante en todo este esquema.

Palabras claves: Metodología de desarrollo de software, equipos de desarrollo de software dispersos geográficamente, herramientas para el soporte de desarrollo de software, características de equipos dispersos geográficamente, control y gestión de proyectos de desarrollo de software distribuidos.

1. INTRODUCCION

1.1 Descripción del Problema

Debido a la falta de una adecuada metodología de trabajo para el desarrollo de software en ambientes virtuales, se evidencia como problema principal a resolver la dificultad en la coordinación de proyectos de software en este tipo de ambientes, esto conlleva a una falta del seguimiento del proyecto de desarrollo software en sus diferentes etapas.

Asimismo, las tareas entre los grupos de trabajo se hace compleja debido a la escasez de herramientas colaborativas que ayuden en la gestión y control de proyectos de software como también al poco conocimiento en la integración de estas herramientas en los procesos de desarrollo de software remotos; por lo tanto la sumatoria de todos estos inconvenientes dificultan que los proyectos de desarrollo de software en ambientes virtuales sean culminados en tiempo y forma.

El desarrollo de software representa hoy en día una industria en creciente evolución donde la exportación de este tipo de servicios se convierte en una actividad de condiciones inmejorables para el país y para las empresas de tecnología; de esta manera el aporte que se espera de esta tesis se encuadra en tres aspectos:

- Social, por la generación de trabajo especializado.
- Económico, por la obtención de ingresos genuinos.
- Tecnológico ya que se mejoran procesos de desarrollo de software en ambientes heterogéneos.

En este contexto el teletrabajo representa una solución acorde para llevar a cabo proyectos de desarrollo de software a distancia involucrando grupos de personas en ambientes virtuales; por lo tanto es importante aplicar una metodología para coordinar, controlar y estimular el correcto funcionamiento de estos grupos humanos a fin de cumplir con los objetivos establecidos en el plan de proyecto de desarrollo de software.

Adicionalmente, hay que tener en cuenta que con la convergencia de las tecnologías de la información y la comunicación (TIC[1]) y el surgimiento de la llamada sociedad de la "información" o "del conocimiento" (SI), el conocimiento y la información se han convertido en un factor determinante de nuestra vida económica, social y cultural. Por lo tanto, es importante considerar que a través de esta metodología se está generando conocimiento aplicable que opera sobre productos software, cuyo mercado está en constante cambio y expansión mejorando de esta manera la relación calidad / costo.

1.2 Propuesta

El diseño de la metodología consistirá en la elaboración de una estructura conformada por 3 capas vinculadas, donde la superior estará formada por el segmento de Control o

1 TIC: Seguiremos la recomendación del Diccionario Panhispánico de Dudas (de la Real Academia Española de la Lengua) de evitar el uso, copiado del inglés, de realizar el plural de las siglas añadiendo al final una -s minúscula, precedida o no de apóstrofe. RAE, 2009. http://buscon.rae.es/dpdl/

Gestión, que se encargará de administrar y coordinar el proyecto de software; luego, en la intermedia, se encuentran las metodologías de desarrollo de software que mejor se adapten al proyecto en ejecución y al desempeño de las tareas en ambientes virtuales, y la inferior estará conformada por el conjunto de herramientas colaborativas que soporten el desarrollo de las capas superiores y el proyecto de desarrollo de software en su totalidad.

Este modelo estará soportado por un conjunto de indicadores que medirán la efectividad de la metodología en ambientes de trabajo virtuales.

1.3 Impacto

Con la aplicación de una metodología adecuada para el desarrollo de software en ambientes virtuales se logrará una mejora sustancial en la coordinación y control de grupos de personas trabajando en entornos heterogéneos. Esto posibilitará la expansión de esta forma de trabajo en factorías de software virtuales o reales impactando en las economías nacionales, desconcentrando y descentralizando la capacidad productiva, permitiendo de esta manera brindar mayores oportunidades de trabajo a especialistas en las TIC de todo el pais.

1.4 Alcance

El diseño de una metodología para el desarrollo de software en ambientes virtuales se aplica exclusivamente a los procesos/tareas que forman parte de cada etapa del ciclo de vida de desarrollo de software, teniendo como forma de trabajo principal al teletrabajo. Además, para que la metodología presentada en esta tesis sea efectiva, deberá apoyarse sobre una o varias metodologías de desarrollo de software que mejor se adapten al modelo planteado.

1.5 Fundamentación

Los orígenes del teletrabajo como se entiende en la actualidad se remontan a la introducción del término telecommuting [Nilles, J., 1976], para referir este tipo de trabajo fuera del lugar habitual.

La aplicación moderna del teletrabajo se puede ver en los Estados Unidos, en donde algunas empresas como Arthur Andersen, IBM y Microsoft, disminuyeron sus costos y aprovecharon las posibilidades que les brindaba la tecnología. La idea de estas organizaciones es que no importa el ámbito en que se desarrollen los trabajos mientras las tareas se cumplan.

Desde principios de la década pasada, el desarrollo de las tecnologías de la información y las telecomunicaciones en los sistemas productivos, se ha constituido en uno de los vectores estructurantes de los cambios sociales más sustantivos para las sociedades occidentales. Este papel del cambio tecnológico, ha producido una profunda transformación en los modelos de organización del trabajo y de la producción y por lo tanto, en las relaciones socio-laborales. En este terreno es donde aparece el concepto del teletrabajo, como término que describe una realidad, multiforme y diversa, de

actividades laborales de trabajo no presencial en las empresas, sustentado sobre las diferentes tecnologías de la información y de la comunicación.

Si bien el trabajo no presencial o a distancia de los centros de trabajo es un hecho tradicionalmente extendido en algunas actividades productivas (como el trabajo a domicilio en el textil o el calzado), el teletrabajo difiere de éstas prácticas por ser susceptible de ser implantado en diversas actividades y en distintos tipos de trabajo, alterando sustancialmente las prácticas y estructuras organizativas de las empresas, así como las condiciones espaciales y temporales de la prestación laboral por parte de los trabajadores.

Con la aparición de las TIC, se amplió la posibilidad de integrar equipos "globalizados", en los cuales el espacio y el tiempo no cuentan. En estos equipos es mucho más importante aún el manejo de las relaciones y el peso de la interdependencia [Abdallah, G. A., 2004]. El Teletrabajo redefine los aspectos del trabajo tradicional, y uno de los principales, es la orientación del trabajo por objetivos y resultados, lo que redunda en una autonomía del individuo.

Es así como a finales de los 80's nace una nueva área de investigación, en plena expansión, la cual cubre los sistemas distribuidos y las redes, y que es denominada Trabajo Colaborativo Asistido por Computadora [Bannon y Schmidt, 1989]. Esta área se interesa fundamentalmente en el diseño y desarrollo de ambientes colaborativos que permiten asistir al trabajo en grupo con el fin de realizar una actividad común y ofrecen una interfaz a un ambiente compartido.

Estos sistemas, también denominados sistemas groupware[2], deben tomar a cargo problemas importantes tales como: el acceso, la distribución y el almacenamiento de la información; la administración y la protección de recursos comunes; el control de acceso y la actualización de la información compartida; la notificación de contribuciones, la interacción, negociación y comunicación entre los participantes; así como su visualización a través de una interfaz apropiada, entre otros.

Gracias a estos sistemas, un grupo de personas puede colaborar, interactuar, negociar y compartir información con el fin de producir de manera coordinada y coherente aquello que antes producían, en forma separada y unían manualmente, a precios de esfuerzos importantes de comunicación y concertación.

En cuanto al diseño de una metodología de trabajo en ambientes virtuales para el desarrollo de software, no existe suficiente información al respecto pues pareciera que no se ha tratado en profundidad esta posibilidad. En este contexto, y acercándose conceptualmente a lo que se plantea en esta tesis, surge un proyecto de investigación nacional en España, titulado "Análisis, diseño e implementación de un centro satélite de

2 Groupware: elemento de software que permite la comunicación, cooperación y la colaboración efectivas en un grupo de agentes activos distribuidos, trabajando de manera coordinada en una tarea común.

teletrabajo" [Ribagorda y Ramos, 1999] donde se pretende el desarrollo completo de un centro de teletrabajo, que incluye:

- Estudio de necesidades
- Diseño del centro (equipamiento, definición e implantación de la red)
- Desarrollo de aplicaciones avanzadas
- Gestión y organización del centro de teletrabajo

1.6 Objetivos

Diseñar una metodología flexible de desarrollo de software para equipos de tareas trabajando en ambientes virtuales

- Investigar las existencias de metodologías que contemplen similares características a la formulada en el presente trabajo

- Evaluar fortalezas y debilidades de las metodologías existentes

- Diseñar una metodología basada en 3 capas que permita una mejor coordinación y seguimiento de proyectos de desarrollo de software en ambientes virtuales

    - Analizar diferentes mecanismos de Control y Gestión de proyectos de software que mejor se adapten a procesos de desarrollo trabajando en ambientes virtuales.

    - Analizar diferentes metodologías de desarrollo de software que mejor se adapten a grupos de trabajos en ambientes virtuales.

    - Definir un conjunto de indicadores que permitan medir la efectividad de la metodología.

    - Utilizar herramientas colaborativas en entornos de desarrollo de software en ambientes virtuales

        - Evaluar algunas de las mejores herramientas colaborativas disponibles en el mercado para ser implementadas.

        - Determinar las herramientas adecuadas según las características del grupo de tarea.

        - Proponer nuevas herramientas colaborativas para un mejor control del trabajo individual.

1.7 Metodología

Se trata de un estudio exploratorio orientada al diseño de una metodología en la que se tendrán en cuenta las siguientes etapas:

Etapa 1: En esta etapa se determinará el estado existente de metodologías aplicadas al desarrollo de software en ambientes virtuales y sobre la aplicación / evolución del teletrabajo en proyectos de desarrollo de software. Para ello se realizará un relevamiento de información por el cual se llevará a cabo una búsqueda documental a través del relevamiento en libros y revistas especializadas, eventos, repositorios de centros de investigación accesibles por Internet y contactos con expertos.

Etapa 2: Una vez determinado el estado actual respecto de las metodologías en uso y la evolución del teletrabajo en proyectos de desarrollo de software, se procederán a estudiar y analizar las técnicas, métodos y mecanismos utilizados para el Control y Gestión de proyectos de desarrollo de software, como así también el análisis y evaluación de las herramientas colaborativas más utilizadas, generando de esta manera, una propuesta de construcción de nuevas herramientas para el desarrollo de software en ambientes virtuales.

Etapa 3: Se elaborará el diseño teórico de la metodología para el desarrollo de software en ambientes virtuales integrando los resultados obtenidos en las etapas previas.

Etapa 4: Se realizará una implementación práctica de la metodología propuesta, evaluando de esta manera la eficacia de la misma.

Etapa 5: Se llevarán a cabo las conclusiones a partir de los logros obtenidos en las etapas anteriores a fin de redactar el informe final de tesis, dejando asentada las futuras líneas de investigación.

1.8 Estructura de la tesis

Esta tesis se estructura en ocho capítulos.

Capítulo 1. Introducción. Aquí se presentan el tema general de la tesis, el problema abordado, la solución propuesta, el impacto, el alcance, la fundamentación, los objetivos y la metodología utilizada para la elaboración del presente trabajo. Además, se describe la estructura de la tesis.

Capítulo 2. Estado del Arte. Se exponen distintas definiciones de reputación según diferentes autores. Se explica los orígenes, importancia y las ventajas y desventajas del teletrabajo. También se detalla los aspectos principales de los ambientes colaborativos, la importancia del trabajo en grupos, las ventajas que ofrece la tecnología para el trabajo en grupos y las condiciones para aplicar la tecnología del groupware. Se realiza una

exposición acerca del desarrollo distribuido de software, las metodologías de desarrollo de software y una descripción de la gestión de proyectos de software.

Capítulo 3. Metodología. Se explica por qué desarrollar una metodología de desarrollo de software. Se detalla cuáles son los requisitos tecnológicos esenciales para lograr un exitoso desarrollo distribuido de software. Se explica la importancia de desarrollar una metodología flexible para entornos de trabajos dispersos geográficamente. Se expone lo trascendental que son las relaciones entre las personas. Se definen un conjunto de indicadores para medir la eficacia de la metodología adoptada.

Capítulo 4. Desarrollo de la metodología propuesta. Capa 3. En este capítulo se detallan las 3 capas que sirven de apoyo para la solución metodológica propuesta, gestión, metodología de desarrollo de software y herramientas.

Capítulo 5. Conclusiones. Se presentan las conclusiones y aportes derivados del trabajo.

Capítulo 6. Futuras líneas de investigación. Se detallan las líneas de investigación futuras sugeridas como ampliación al presente trabajo.

Capítulo 7. Citas y referencias bibliográficas. Se nombran los trabajos que aportaron a elaboración de esta tesis.

Capítulo 8. Anexo. Se expone el trabajo "Utilizando un proceso de desarrollo ágil en un proyecto offshore" de Martin Fowler[3] que sirvió como documento inspirador para la realización de la presente tesis.

3 Martin Fowler: Orador, Consultor y escritor de diversos artículos y libros referidos al Desarrollo de Software. http://www.martinfowler.com/aboutMe.html

2. ESTADO DEL ARTE

2.1 El Teletrabajo

2.1.1. Importancia del Teletrabajo

Desde el siglo pasado estamos atravesando por una nueva revolución, que en los umbrales del nuevo siglo XXI es imposible dejar pasar inadvertida. Se trata de una revolución científica y tecnológica donde la telemática es la principal protagonista.

El Informe Nora - Minc [Nora, S. Minc. A., 1978] fue en su pronóstico un nivel de macroanálisis. Es decir, con la introducción de dicho neologismo que implica la convergencia entre informática y telecomunicaciones, los ministros franceses dieron cuenta de un hecho inobjetable: la informatización de la sociedad crea una nueva forma de relacionarse con las cosas en el ámbito económico, político y social; dicho de otro modo, tiene la facultad de transformar la realidad. Esta transformación no se desarrolla tanto a partir del esfuerzo físico del hombre sino sustancialmente a partir del esfuerzo intelectual. Hete aquí, la Sociedad de la Información y del Conocimiento (en adelante, SI).

Este cambio tecnológico ha propiciado la aparición de una novedosa modalidad laboral: el "Teletrabajo". El Teletrabajo es impensable fuera de la constitución de la SI ya que implica realizar una actividad específica remunerada, fuera del lugar físico de la empresa, mediante el uso intensivo de las Tecnologías de la Información y la Comunicación (en adelante, TIC).

Las ventajas que reciben los distintos sectores involucrados en esta modalidad, son bien significativas pero no más importantes que sus debilidades. En líneas generales, la argumentación favorable versa sobre el ahorro de infraestructura, aumento de productividad y mayor flexibilidad.

Actualmente vivimos inmersos en la globalización y dominados por las TIC, por lo que es menester aprovechar al máximo los recursos que participan de los intercambios trasnacionales.

El Teletrabajo ofrece esta posibilidad. Mientras Europa padece un déficit de especialistas en el uso de las TIC, Latinoamérica ofrece profesionales al mundo. Argentina es un claro ejemplo de ello.

Es aquí donde se presenta el mayor desafío. El Teletrabajo es una instancia beneficiosa y superadora, siempre que haya una voluntad política de asegurarla a través de marcos legales

propios, programas de educación y capacitación continua, políticas de difusión y desarrollo, y estrategias de intercambio entre empresas y teletrabajadores del mundo.

Es necesario aclarar, que el Teletrabajo no es ni será el único modelo de relación laboral, pero sí, una oportunidad que ofrece la tecnología y que es plausible de ser bien aprovechada. El Teletrabajo es un modelo que está en continua evolución y adhesión, a medida que se comprende su naturaleza y se confía en él. La confianza, por supuesto, se construye y se logra a partir de un análisis que contemple todas sus ventajas y desventajas, aciertos y contradicciones, fortalezas y debilidades.

2.1.2. Introducción

Desde la última década del siglo XX, la velocidad de las transformaciones tecnológicas es cada día mayor e impacta a todos los órdenes de la vida económica, política y cultural. Las relaciones laborales no escapan a estos cambios y surgen nuevas formas de trabajo que suelen considerarse y denominarse como modalidades flexibles.

Estas flexibilizaciones, muchas veces "de hecho" y violatorias de las normas laborales vigentes, implican graves riesgos para los trabajadores, por el desconocimiento de sus derechos. Desde otro ángulo, la globalización, ha generalizado también el fenómeno de la desocupación. Este gravísimo problema que padece el mundo postmoderno, es campo propicio para ello, ya que estos progresos no siempre se implementan desde una visión humanista, que privilegie el Desarrollo Humano y brinde nuevas esperanzas para las próximas generaciones. Por el contrario, se precariza el trabajo para reducir sus costos.

El nuevo contexto que ofrece la SI y el desarrollo de las TIC incide en los ámbitos económico, político, social y cultural. Los cambios acontecidos en el campo laboral son un claro referente de las transformaciones propiciada por la denominada "economía digital".

Por un lado, emergen nuevos empleos y perfiles profesionales que requieren el dominio de estas tecnologías. Por otro, surge un nuevo modelo de búsqueda de trabajo: la RED se convierte en el nexo coordinante entre oferentes y demandantes.

La incorporación de las TIC en el ámbito empresarial puede resultar un elemento clave para impulsar el crecimiento económico y lograr una mayor creación de empleo. Es por ello, que en favor de una comunicación mundializada, el papel de los países en el orden mundial dependen de su presencia en la Sociedad de la Información. El Teletrabajo es una oportunidad que brinda la Economía Digital y que es posible de ser optimizada y aprovechada.

Este papel del cambio tecnológico ha producido una profunda transformación en modelos de organización del trabajo y de la producción y, por lo tanto, en las relaciones socio-

laborales. En este terreno es donde aparece el concepto del teletrabajo, como término que describe una realidad, multiforme y diversa, de actividades laborales de trabajo no presencial en las empresas, sustentado sobre las diferentes tecnologías de la información y de la comunicación.

Si bien el trabajo no presencial o a distancia de los centros de trabajo es un hecho tradicionalmente extendido en algunas actividades productivas (como el trabajo a domicilio en el textil o el calzado), el teletrabajo difiere de éstas prácticas por ser susceptible de ser implantado en diversas actividades y en distintos tipos de trabajo, alterando sustancialmente las prácticas y estructuras organizativas de las empresas, así como las condiciones espaciales y temporales de la prestación laboral por parte de los trabajadores.

En el contexto de los países más desarrollados (en varios de los cuales el número de teletrabajadores es ya considerable) su tendencia, es la de extenderse como forma complementaria a la organización central de las empresas, apareciendo como un proceso de las estrategias empresariales de externalización de actividades, dentro de las políticas de desconcentración y descentralización productiva, tanto en los sectores industriales como en los servicios.

2.1.3. Definición de Teletrabajo

Etimológicamente el término Teletrabajo procede de la unión de la palabra griega tele, usada como prefijo que significa "lejos", y de trabajo, que es la acción de trabajar; palabra latina (tripaliare de tripálium, instrumento de tortura) que significa: realizar una acción física o intelectual continuada, con esfuerzo. De aquí se deduce la primera aproximación al término: “un trabajo lejos, es decir, a distancia”.

El término Teletrabajo apareció por primera vez en Estados Unidos en 1973 de la mano de Jack Nilles, físico y antiguo investigador de la NASA. Eran los tiempos de la crisis del petróleo y lo que se pretendía era evitar los desplazamientos para ahorrar consumo y reducirla contaminación. Teletrabajo, aparece entonces, como la traducción de “Telecommuting”; se trataba de "enviar el trabajo al trabajador en lugar del trabajador al trabajo".

Este concepto fue tomado con éxito en EE.UU., hasta incluso llegó a eclipsar el término “Teleworking”, propio de la Unión Europea.

La diferencia que se presenta entre ambos términos es que éste último, pone mayor énfasis en el trabajo a distancia utilizando intensivamente las Tecnologías de la Información y la Comunicación.

Esta situación llevó a la imposibilidad de normar el concepto y a que numerosas veces con la palabra Teletrabajo se definan conceptos diferentes. Por ejemplo, una persona que esté utilizando un teléfono estaría teletrabajando para Nilles pero no lo sería según la óptica Europea.

Tampoco debe relacionarse el "Teletrabajo" como trabajo a domicilio, ni caer en la percepción errónea de que un teletrabajador es una profesión y mucho menos una revolución tecnológica.

El Teletrabajo es una forma alternativa de organización del trabajo, es un elemento más dentro de un proceso más amplio. Para analizar y entender el Teletrabajo, es necesario encuadrarlo dentro de la Sociedad de la Información (SI).

En este sentido es muy importante rescatar el concepto de modo de desarrollo [Castells, M., 1995]. La tendencia que se viene desarrollando desde fines de la década del '70 y comienzo de los '80, es la re-estructuración del capitalismo en base al procesamiento y producción de conocimiento; de ahí parte la hipótesis del advenimiento del modo de desarrollo Informacional.

El modo de desarrollo implica cómo están organizadas las relaciones técnicas de producción, y en esta era es en base a la aplicación de microprocesos de información. La particularidad del paradigma tecnológico es que no sólo la información deviene materia y proceso a la vez, sino que también revoluciona todos los sistemas productivos. En este contexto surge el Teletrabajo; es sintomático de los cambios a todo nivel (económico, político y social) que genera la Sociedad de la Información. El Teletrabajo es entonces posibilitado gracias a la interrelación de dos componentes: la aplicación laboral de las tecnologías de la información y la existencia de una infraestructura de telecomunicaciones razonablemente avanzada.

Entonces podríamos comprender el Teletrabajo dentro de tres lógicas de inversión que hacen al Proyecto de una SI:

- La definición de Teletrabajo propuesta por J. Nilles [1976]: "que el trabajo va hacia el trabajador y no a la inversa", expresa la inversión de un esquema de relación laboral consolidado por el modo de desarrollo industrial capitalista.
- Inversión de la lógica nacional-global: el Teletrabajo, por su propia naturaleza, es transregional, transnacional y transcontinental. Las transacciones de Teletrabajo suceden muy rápidamente en el marco de la mundialización y de la homogeneización de lógicas infocomunicacionales. El Teletrabajo no tiene fronteras.

- Inversión de la lógica divergencia/convergencia: sólo se teletrabaja cuando existe una conexión telemática entre el equipo remoto y el de la organización, por la que se establece una línea de interactividad con los sistemas de información de la empresa y con el resto de los trabajadores. Esto permite al teletrabajador participar, con las limitaciones del puesto que ocupa, en el conocimiento de la organización. Cual fuera el caso; casa particular, telecentros o cualquier otro tipo de oficina que dispone de equipamiento informático y de telecomunicaciones desde donde trabajar, el elemento común al concepto del Teletrabajo es el uso de ordenadores y telecomunicaciones para cambiar la modalidad y el ámbito laboral.

Es preciso reiterar que en la actualidad existen multitud de definiciones del término Teletrabajo y de ahí que sea muy difícil cuantificar y/o conocer su implantación real.

Además, la inestabilidad del concepto de Teletrabajo se debe al hecho de que éste depende intrínsicamente de los cambios tecnológicos, por lo que se ve determinado a evolucionar junto con ellos.

A pesar de lo descripto, se puede extraer tres características clave que aparecen en la mayoría de las definiciones sobre Teletrabajo. Ellas son: organización, localización y tecnología.

2.1.4. Orígenes del Teletrabajo

Para trazar un recorrido histórico en cuanto a la aplicación del Teletrabajo debemos situarnos en la década del 70.

Como se mencionó anteriormente, en 1973, Jack Nilles, presidente de la consultora JALA Assocites (además de físico de la NASA) introduce el término telecommuting, es decir, teledesplazamiento, en un marco de crisis energéticas y grandes problemas de congestión del tráfico que se dan en grandes ciudades como Los Ángeles. Es así como en el estado de California se dictan leyes que obligan a las empresas a implementar programas de Teletrabajo para reducir el tráfico y evitar la contaminación. El Teletrabajo comienza a ser aplicado a grandes grupos de trabajadores, midiéndose los resultados y elaborando conclusiones. Estas primeras experiencias servirán de modelos para posteriores aplicaciones.

Hacia los años 80, E.E.U.U. aplica proyectos piloto de Teletrabajo, mucho más que Europa. Es en este momento cuando la Comunidad Económica Europea comienza a investigar las posibilidades del Teletrabajo para el desarrollo rural, las implicaciones sociales (la protección de los teletrabajadores), y los aspectos tecnológicos dentro de los primeros programas de investigación y desarrollo tecnológico.

La evolución del Teletrabajo no es homogénea, siendo los países anglosajones los que encabezan las líneas de desarrollo, especialmente y no casualmente, E.E.U.U. En esos años, E.E.U.U. era dueña de una ferviente y hegemónica política neoliberal que había llevado a la liberación del sector de las telecomunicaciones, a una descentralización de las empresas y al imperio de las multinacionales. Mientras tanto, en Europa primaba las políticas públicas en la evolución del Teletrabajo, ésta era considerada una herramienta propicia para el desarrollo regional y además, para la búsqueda de nuevas fórmulas de trabajo que dieran empleo a la gran cantidad de desempleados que existía.

La crisis económica mundial de mitad de los años 90, hace que las empresas busquen al Teletrabajo como una fórmula para reducir costos ante la caída de la demanda producida. Es así como esta modalidad se va asentando en el mundo.

La Unión Europea, consciente del papel del Teletrabajo en la sociedad, y como una de las líneas del Libro Blanco sobre el crecimiento, la competitividad y la ocupación de Delors [Comisión Europea, 1993 a], formula unas Acciones de Estímulo al Teletrabajo. Dicho libro, presentado en Bruselas en diciembre de 1993, es el primero de una serie de informes en relación a las posibilidades que ofrece la SI.

El Libro Blanco manifiesta una preferencia por el fomento y la flexibilización de las formas de empleo para paliar los persistentes problemas de paro en la Unión Europea. Este documento ponía la transición hacia una sociedad de la información en el centro de las políticas de la Unión Europea en cuanto a crecimiento y competitividad.

El Teletrabajo es la modalidad que encierra en la práctica de ambas políticas: por un lado es flexible, tanto en tiempo como en lugar, y además, ofrece una gama más amplia de posibilidades de empleo en zonas rurales y apartadas. Las bases para el desarrollo del Teletrabajo en Europa son los programas de investigación de la Unión Europea sobre comunicaciones avanzadas (RACE) y aplicaciones telemáticas. Tales programas han permitido que los nuevos sistemas digitales de comunicación estén ya implantados en la mayoría de los estados integrantes de la Unión Europea; que sean compatibles entre si, que estén preparados para evolucionar a superautopistas más rápidas y que se puedan usar de manera coherente para generar crecimiento y empleo en el tejido empresarial europeo.

En Corfú[4], el 24 y 25 de junio de 1994, el industrial Martin Bangemann presentó al Consejo Europeo el informe conocido como “Europa y la sociedad global de la información” [Comisión Europea, 1994 b]. Este documento se basa en la exposición de medidas específicas que debía estudiar la Comunidad Europea para la adopción y viabilidad del establecimiento de infraestructura de la SI.

4 Isla griega en el mar Jónico, entre la costa noroeste del Epiro griego y al sur del Epiro albanés.

En un apartado del informe, se manifiestan diez aplicaciones experimentales de la SI, siendo el Teletrabajo aquella que encabeza la lista. De este modo aparece la primera aproximación a una forma de trabajo consecuente con el desarrollo informacional. El informe Bangemann, propone fomentar el Teletrabajo en casa y en oficinas satélite, de manera que no sea necesario desplazarse largas distancias para ir al trabajo. Además, se recomienda un calendario de objetivos:

- A fines de 1995, veinte ciudades europeas han de tener 20.000 teletrabajadores en centros pilotos.
- A finales de 1996 debe estar teletrabajando el 2 % de la masa laboral europea.
- Deberá de haber 10 millones de teletrabajadores en Europa a final de la década.

Por otra parte, se mencionan los problemas a resolver: dificultades de promoción social, legislación laboral y seguridad social.

Hacia 1996 la Comisión Europea publica el Libro Verde Prioridad para las personas [Comisión Europea, 1996 c], que a diferencia de su antecesor, los lineamientos están atravesados en torno a la problemática del empleo. Si bien en este texto no se habla específicamente del Teletrabajo, se le hace referencia implícita al tratar cuestiones como: la capacitación laboral de los ciudadanos en torno a las TIC para evitar desempleos a largo plazo, fomentar el conocimiento de las nuevas formas de organización del trabajo, establecer directivas para explotar el potencial competitivo de las empresas, y en definitiva, la preocupación por la cohesión social.

Durante la segunda mitad de los años 90, las empresas emplean el Teletrabajo para lograr un contacto más cercano de sus clientes. Hay un decisivo impulso al Teletrabajo por parte de las administraciones públicas, sobre todo en la Unión Europea, difundiéndose los resultados de experiencias y logrando una mayor integración entre el Teletrabajo y la sociedad. En EE UU, por su parte, el desarrollo iniciado antes, es mucho mayor que en Europa; la explosión de Internet fue un fenómeno que contribuyó a tal desarrollo.

De este panorama planteado en relación a la aparición y evolución del Teletrabajo, se puede sintetizar que el nacimiento del mismo se produce cuando la convergencia de la informática y la tecnología de las comunicaciones posibilitaron la descentralización de algunos tipos de trabajo que tenían que ver con el procesamiento electrónico de la información. Su mayor apropiación, ejercicio y desarrollo, está en manos de quienes cuentan con la infraestructura necesaria y régimen de políticas en el área de las telecomunicaciones.

2.1.5. Ventajas y desventajas del Teletrabajo

Como ya lo preludiaba El informe Nora-Minc [Nora S, Minc, A, 1980] al tratar la informatización en la gestión de empresas y servicios, la adopción del Teletrabajo implica

un pasaje de una organización empresarial de dirección muy centralizada, jerárquica y vertical, a una organización plana, democrática y descentralizada. En este marco, es necesario evaluar los pro y contra de la implantación del Teletrabajo, antes de caer en la fatalidad tecnológica de creerla la única posibilidad laboral en la SI, ni tampoco, condenarla arbitrariamente. Esta nueva modalidad, como todo cambio que se precie de tal, genera una serie de ventajas e inconvenientes. En este caso, las mismas serán abordadas desde la integración de tres perspectivas: la del trabajador, la de la empresa y la de la sociedad.

Uno de las principales ventajas del Teletrabajo para el trabajador es el aumento de la flexibilidad del trabajo, tanto de horarios como del orden de las tareas a afrontar, lo cual le permite distribuir el tiempo de la forma que crea más conveniente, facilitando un mayor autocontrol y productividad.

Por otro lado, al no estar limitado ni por una localización geográfica, ni por horarios, el Teletrabajo abre una abanico de oportunidades laborales.

Otra de las ventajas percibidas de este sistema es que permite una mayor adecuación persona-trabajo. Es decir, el teletrabajador podrá aceptar aquellas tareas que mejor se adecuen a sus características, formación, capacidad y situación personal y profesional, sin verse limitado por otras consideraciones.

Uno de los argumentos más fuerte a favor del Teletrabajo es la posibilidad de combinar de forma más satisfactoria la vida laboral y la familiar. El hecho de trabajar desde su propio domicilio redunda en mayor dedicación a la familia y tareas de la casa.

Por último, la reducción de los desplazamientos y el stress generado por la vida cotidiana de la oficina, son consideradas ventajas propias de esta modalidad de trabajo.

Para la organización, la ventaja más evidente es la reducción de costos, sobre todo los de alquiler de inmuebles, de mobiliario, de transporte, etc., como así también el aumento de productividad en aquellos trabajadores que mejor se adaptan al Teletrabajo.

Como contraparte del aumento de oportunidades para el trabajador, el Teletrabajo ofrece a las empresas, un aumento de las posibilidades de contratación. Además, las contrataciones se vuelven más flexibles ya que el trabajo puede quedar determinado por el control de objetivos y resultados.

Es decir, en este sistema no se juzgan las horas que el trabajador está en su puesto, sino la calidad del trabajo realizado y el cumplimiento de los plazos de entregas. Por otra parte, el Teletrabajo reduce el absentismo laboral y permite sortear conflictos laborales como las huelgas.

Por su parte, la sociedad se ve beneficiada en la reducción de la congestión de tráfico y de la consiguiente polución; en la promoción del desarrollo de las zonas rurales, aisladas o deprimidas; en la posibilidad de integrar en el circuito laboral a minusválidos o aquellos sectores de la población que por motivos personales no pudieran desplazarse hasta el puesto de trabajo. Por último, otra de las ventajas que presenta, es la mejora en la colaboración entre áreas, por el uso continuo y extensivo de las TIC. Esto permite personas de todo el mundo, casi a tiempo real, colaboren en el desarrollo de un mismo proyecto.

Pero de todas maneras es necesario enumerar los elementos negativos o desventajas que trae la modalidad del Teletrabajo.

Para las personas, una de las desventajas puede ser la falta de interacción diaria entre compañeros y de vida social, lo que puede provocar pequeñas crisis de aislamiento o soledad. Además, el teletrabajador al perder el contacto presencial con la jerarquía empresarial, enfrenta el temor de no ser promovido y experimentar la sensación de estancamiento de su carrera. En este sentido, las medidas iniciales de sensibilización e información (que tanta mención hace los Libro Verde europeos) y el mantener un continuo contacto y preocupación entre trabajadores y directivos contribuirá a disminuir esta percepción. Por otra parte, se puede argumentar que el menor contacto cara a cara facilita el que las promociones se basen menos en factores como el atractivo físico, la raza o cualquier aspecto que pueda generar selectividad o marginación.

Otro factor a considerar es que no todos los trabajadores son capaces de articular trabajo y familia, lo que puede llevar a desatenderla, o a la saturación de trabajo provocando una baja en el rendimiento laboral. Entre los efectos colaterales del Teletrabajo se pueden mencionar los relacionados a la salud: jaquecas, cansancio visual y dolores musculares o posturales.
Otro aspecto fundamental es la falta de legislación específica para el teletrabajador. De hecho, los modelos de contrato existentes son elaborados por las empresas siguiendo sus propios criterios, lo cual repercute negativamente en la protección social del teletrabajador [Nicolosi, A., 2006].

Para las organizaciones, una de las más evidentes desventajas es el alto coste inicial de los equipos y de la infraestructura con la que es necesario dotar a la empresa. Otras desventajas que implica son: acostumbrar a la directiva y empleados al sistema de dirección por objetivos, ya que no siempre es posible controlar las horas que el teletrabajador dedica a su tarea; el tema de la seguridad en las comunicaciones y la confidencialidad, ya que como la información relativa a la empresa circula por Internet es necesario coartar el libre acceso a la misma. Asimismo, se presenta la incapacidad de los directivos para aceptar el nuevo tipo de relación laboral, pues la falta de contacto con sus empleados lleva a la sensación de pérdida de las señales visibles de su poder o de su status.

A modo de conclusión puede decirse que es necesario ver el Teletrabajo como un medio y no como un fin en sí mismo; como una posibilidad que trae sus beneficios y sus inconvenientes. Pero no hay que arriesgarse a generalizar, ya que en el fondo es la persona que está detrás, la que puede valer para el trabajo o no.

La visión positiva de la SI santifica la incorporación de las TIC al ámbito empresarial, señalando singular cantidad de logros y beneficios: competitividad, ahorro de costos, generación de nuevos puestos y oportunidades de trabajo, etc. Sin embargo, existen posturas que ven en las nuevas tecnologías un argumento que no tiene verificación en el campo real [Castellón, L., 2001].

Dentro de esta perspectiva, las TIC son consideradas fuentes destructoras de empleo y generadoras de exclusión [Castellón, L., 2001].

En primera medida porque la innovación tecnológica implica un menor uso del factor trabajo y en segunda instancia porque supone dirigir la demanda empresarial hacia sólo perfiles profesionales cualificados, aptos para el desempeño en el mundo en red.

Aquí nos topamos con el problema del acceso y dado que el Teletrabajo implica un cambio de mentalidad, no debe limitarse este concepto a la mera adquisición de la herramienta [Castellón, L., 2001]. La problemática del acceso aborda otras esferas del individuo como ser por ejemplo, la competencia cultural que exige el uso de las tecnologías, es decir, el conocimiento específico y operativo que ellas requieren para ser aprovechadas. La brecha generacional está íntimamente relacionada a lo anterior. Sin caer en prejuicios, las jóvenes generaciones son las mejores predispuestas a comprender racionalmente las nuevas tecnologías, a diferencia de las pasadas, se formaron y estructuraron en torno a medios artesanales de trabajo y convivencia. La cuestión de géneros (que será tratada posteriormente) tampoco queda exenta del problema del acceso.

Por último, la más visible de las barreras: el factor económico. Fuera de la gestión de los telecentros, sólo accederán al Teletrabajo quienes (empresas e individuos) cuenten con lo recursos económicos suficientes para invertir en software y hardware.

En síntesis, lo expuesto demuestra que para una implementación eficaz del Teletrabajo, deben crearse programas que definan estrategias de aplicación, a partir de la superación de la problemática del acceso en términos materiales.

2.2 Ambientes Colaborativos

2.2.1. Introducción

El progreso y desarrollo tecnológico de las últimas décadas ha provocado un cambio en la concepción y estructuración de los procesos organizativos y de las relaciones que se establecen entre los miembros de una organización. El teletrabajo, la educación a distancia, el servicio al cliente on-line, las transacciones electrónicas, son ejemplos de relaciones cotidianas (obrero-patrón, cliente-empresa y empresa-empresa) que han sido innovadas y mejoradas mediante la incorporación de un componente básico: el tecnológico. En este último, convergen avances significativos en informática, aplicaciones de redes, telecomunicaciones y audiovisuales, todos ellos ofreciendo una variada gama de técnicas, herramientas computacionales y aplicaciones automatizadas, que por sus características "...permiten ser empleadas para corregir formas, normas o modelos alternativos en los procesos productivos, de organización de trabajo, de prestación de servicios y, en consecuencia, de estructuración social en general" [Aguadero, 1997, p. 11].

En este nuevo orden tecnológico se le da un papel preponderante a aspectos, tales como: interacción, comunicación, transmisión y comportamiento de la información. La conjunción de dichos aspectos con el uso intensivo de las tecnologías informáticas y de telecomunicación, es lo que ha propiciado el surgimiento de lo que se denomina Sociedad de Información, cuya característica principal es su articulación en redes [Aguadero, 1997, p. 19].

La formación de grupos, cuya manifestación en la sociedad actual se ha visto reforzada por la utilización de la tecnología de redes como vía para propiciar la colaboración y el compartimiento de ideas y conocimiento.

En los grupos se observa, por lo general: Una misión y objetivos que cumplir, relaciones de coordinación y cooperación, niveles de jerarquía, asignación de tareas, lineamientos para el comportamiento (derechos y deberes), etc. Entre los aspectos que revisten mayor importancia cuando se trabaja en forma grupal están: contar con una visión común, que canalice los esfuerzos individuales y ofrezca a sus integrantes un sentido de pertenencia, disponer de canales y métodos de comunicación eficientes y la capacidad de compartir información. Actualmente con el desarrollo y auge de las tecnologías de información y comunicación se han diseñado herramientas computacionales que pretenden facilitar dichos aspectos y, al mismo tiempo, reducir los problemas comunes que tiene el trabajo en grupo.

El estudio de una alternativa distinta para trabajar en forma grupal, recibiendo los beneficios de las tecnologías de información y comunicación y la determinación de los factores de administración y organización que se deben considerar para la implantación de la misma, es de vital importancia para que el trabajo en grupo sea efectivo y aprovechado al

máximo. En este contexto podemos mencionar el Trabajo en Grupo Colaborativo, también conocido como Groupware, que permite que un grupo de personas pueda colaborar, interactuar, negociar y compartir información con el fin de producir de manera coordinada y coherente aquello que antes producían, en forma separada y unían manualmente, a precios de esfuerzos importantes de comunicación y concertación.

2.2.2. Definición de Trabajo Colaborativo o Groupware

Se define el Groupware o Trabajo Colaborativo como la tecnología usada para comunicar, cooperar, coordinar, resolver problemas, competir y negociar, facilitando el trabajo de grupos a través de redes de computadoras y los servicios inherentes en las mismas, tales como: e-mail, transferencia electrónica de archivos, grupos de noticias, conversaciones electrónicas, hipertextos, grupos de discusión, etc. [Brinck, 1999]

Las tecnologías de trabajo colaborativo se categorizan en dos dimensiones [Brinck, 1999]:

1. Usuarios trabajando juntos al mismo tiempo (Groupware Sincrónico) o en tiempos diferentes (Groupware Asincrónico).
2. Usuarios trabajando juntos en el mismo lugar (cara a cara) o en sitios diferentes (a distancia).

2.2.3. El Trabajo en Grupo

El Trabajo en Grupo se define como el proceso mediante el cual un conjunto de individuos realizan actividades relacionadas con la finalidad de lograr un objetivo específico, cumplir una meta o compartir una ideología común. Dichas actividades implican una acción colectiva y la realización de trabajos colaborativos. Según la teoría más reciente el trabajo colaborativo persigue *"...el desarrollo de conocimiento compartido, la aceleración de los flujos de información, la coordinación de los flujos de recursos para producir economías de costos y tiempos*" [Díaz, Angel, 2000, p 10].

Al trabajar en grupo se observan por lo general los siguientes aspectos: una coordinación general, asignación de tareas en forma individual, niveles de jerarquía interna, definición de lineamientos de comportamiento y de beneficios por pertenecer al grupo, derechos y deberes, objetivos a lograr en forma colectiva. El aspecto que tiene mayor importancia es el hecho de contar con un objetivo común que canalice los esfuerzos individuales y les ofrezca un sentido de pertenencia que fomente la unión. De no contar con objetivos bien definidos, es probable que los esfuerzos individuales se diluyan bajo la presión de intereses personales o intereses distintos a los del grupo en sí, trayendo como consecuencia que no se logre el propósito para el cual el grupo fue conformado.

La idea de conformar equipos de trabajo proviene de la necesidad que tienen las organizaciones de obtener resultados, producto del consenso grupal y de la revisión

exhaustiva bajo perspectivas distintas, que aseguren la calidad, confiabilidad y exactitud en las soluciones planteadas, productos obtenidos o decisiones elegidas. A su vez, los miembros del grupo tienen la oportunidad de aprender tomando en consideración: otros puntos de vista, maneras distintas de hacer las cosas, interpretaciones diferentes de conceptos, experiencia de otros y como solucionaron problemas similares.

El hecho de que exista un aprendizaje individual propicia que los grupos aprendan y por ende la organización aprenda. Esto crea los mecanismos de aprendizaje que permiten la creación de un conocimiento organizacional o conocimiento detallado de las características específicas de los procesos que maneja la empresa conocido como "Know-How". Es recomendable que la organización proporcione la estructura organizativa y defina las estrategias que propicien el trabajo en grupo y evitando así, las actividades aisladas o personalizadas que no tienen relación alguna con las ideas rectoras de la organización. A continuación se presentan algunos lineamientos iniciales [Hernández Arias, A., 2002]:

- En cuanto a la conceptualización de trabajo en grupo se recomienda utilizar técnicas de liderazgo para fomentar y motivar el trabajo individual en pos del trabajo grupal y tomar en cuenta los comportamientos y habilidades de los miembros asignados. Estas condiciones propician la comunicación efectiva entre los miembros del equipo de trabajo y permiten definir en consenso las estrategias para el manejo de conflictos, elemento inevitable cuando se trabaja en ambientes grupales. Por último, realizar evaluaciones de resultados en función de los objetivos inicialmente planteados para ejercer las acciones correctivas.

- En cuanto a la operatividad del trabajo en grupo es conveniente definir claramente la razón para la cual el equipo de trabajo fue creado: para un proceso de revisión o mejora, para la toma de decisiones o para el diseño o desarrollo de un proyecto. De acuerdo con la razón se determinan las estrategias de trabajo, las actividades requeridas, la metodología a seguir y se seleccionan las herramientas que faciliten y agilicen las actividades grupales.

2.2.4. Ventajas

Si se logra una interdependencia óptima entre propósitos, sistemas y equipos disponibles, el trabajo grupal propicia un ambiente para la comunicación y discusión productiva. Se crea sinergia al aprovechar el conocimiento y experiencia de los miembros, según su área de especialización y los diversos enfoques o puntos de vista, y se logra así una visión completa del estudio a realizar mejorando la calidad de las decisiones y de los productos obtenidos.

Se presenta la posibilidad de realizar procesos en paralelo en función de la división del equipo en comisiones o subgrupos, que permiten aligerar la presentación de la solución, producto u opinión según la situación para la cual trabajan.

La obtención de resultados realizada por consenso grupal asegura la calidad, confiabilidad y exactitud en las ideas y soluciones planteadas al extraer el máximo provecho de las capacidades individuales para beneficio del grupo.

2.2.5. Desventajas

El trabajo en grupo puede generar retraso en la presentación de soluciones debido a la falta de orientación, de propósitos claros y definición de lineamientos. Lo anterior puede traer como consecuencia el enfrentamiento y discusiones no relacionadas con los objetivos planteados.

La falta de recursos, materiales y equipos para la realización del trabajo propicia un ambiente organizacional donde no se cuenta con oportunidades para aplicar los resultados.

El comportamiento de los miembros del grupo puede afectar la productividad del equipo: arrogancia, dificultad para decir no sé, juicios automáticos y sin base, percepción de que es difícil o imposible el trabajo que se hace, inhabilidad para manejar errores, apatía, impaciencia o poca tolerancia, entre otros.

2.2.6. Ventajas que ofrece la Tecnología para el Trabajo en Grupo

En forma general, la tecnología ofrece innumerables ventajas en todos los ámbitos del saber, una de las más importantes es el de considerarse como factor multiplicador, es decir, *"...el potencial de transformación que conlleva la aplicación de la misma. Dicho de otra forma: es el número de veces que una tecnología determinada es capaz de mejorar el proceso o la función a la que se aplica*" [Aguadero, 1997, p. 43]. En relación al Trabajo en Grupo, la tecnología brinda ventajas adicionales, entre las cuales se pueden numerar las siguientes [Aguadero, 1997]:

1. Facilita el acceso a recursos e información de proyectos relacionados con el área de estudio, ya sea bajo la modalidad de hipertexto o bajo la modalidad de grupos de noticias.
2. Permite compartir experiencias exitosas acerca de nuevas técnicas para el mejoramiento de los procesos en la organización, promoviendo la participación de los miembros y de personas interesadas por la facilidad de acceso a los recursos ofrecidos (investigaciones, bases de datos, papeles de trabajo, enlaces a otros recursos a través de Internet, contacto con personas que trabajen en proyectos similares) y la formación de grupos con intereses comunes.
3. Permite y facilita la comunicación, la hace más rápida, clara y persuasiva.
4. Reduce costos y tiempo de transporte de los miembros del grupo hacia espacios físicos específicos.

5. Facilita la resolución de problemas grupales al reducir el tiempo y costo requerido para coordinar el trabajo. Esto se logra a través de calendarios electrónicos los cuales permiten la planificación, gerencia de proyectos y coordinación de los miembros de un grupo por medio de funciones de detección de conflictos de planificación, determinación de horarios de disponibilidad y localización de personas.
6. Permite nuevos modos de comunicación, tales como: intercambios anónimos e interacciones estructuradas a través de software de reuniones a distancia, chateo y videoconferencias.

2.2.7. Dificultades que se pueden presentar al utilizar tecnologías para el Trabajo en Grupo

1. Diferencias en los niveles de experiencia y conocimiento de tecnologías por parte de los miembros del equipo de trabajo. Esto puede traer como consecuencia que algunos miembros tengan dificultades para utilizar y entender ciertas tecnologías.
2. La dinámica del trabajo en equipo es demasiado variable y es probable que las tecnologías no se adapten a cambios constantes en las estrategias de trabajo.
3. Se requiere mayor cantidad de tiempo para incluir el componente tecnológico como parte del trabajo en grupo (pruebas iniciales e inducción, capacitación y formación de los miembros del equipo de trabajo).
4. Cada grupo de trabajo es diferente y su comportamiento es influenciado por las condiciones de trabajo específicas para el período en el cual trabaja, esto quiere decir, que el éxito en el uso de tecnologías para un grupo puede ser un fracaso para otro.

2.2.8. Condiciones para aplicar la Tecnología del Groupware

1. **En cuanto al recurso humano**. Al implantar la Tecnología del Trabajo Colaborativo una de las condiciones principales es facilitar los procesos de interacción entre los miembros del grupo. La interacción permite establecer el diálogo, mediante el cual la organización puede obtener información valiosa para modificar y mejorar sus procesos internos. La tecnología permite establecer nuevas formas de interacción y *"...en la medida que la interacción se automatice y sea cada vez más directa - con el uso de Internet, Extranet y otros sistemas integrados- aumentará su relación eficacia - eficiencia*" [Macchia, 2000, p. 17].

   Adicionalmente a los procesos de interacción, existen diferentes grados de implicación y compromiso que se establecen entre los miembros del grupo. Cada participante se forma una percepción de los objetivos del grupo y la importancia de su participación para la consecución de los mismos permitiéndole identificarse en forma positiva de acuerdo a sus valores y actitudes. De esta manera, entra en un diálogo productivo a beneficio del grupo y de los objetivos que se persiguen. El grado de compromiso con la comunidad propicia el desarrollo de foros, eventos

donde el grupo de trabajo se relaciona en parte con la organización y también con otros que comparten intereses comunes. A través de dichas actividades cada participante se convierte en promotor del grupo, compartiendo con los demás su experiencia positiva. Así la fase de implantación de una propuesta de trabajo colaborativo, utilizando tecnologías de información y comunicación, a de estar ligada a una nueva cultura de uso de nuevas modalidades de trabajo incorporando herramientas computacionales que permitan innovar y mejorar los procesos de trabajo grupal. Se recomienda comenzar con una fase de inducción, capacitación y entrenamiento, fomentar la participación de las personas involucradas en el proyecto y prestar atención a las sugerencias y propuestas de los usuarios involucrados.

2. **En cuanto a la tecnología**. Basándose en el modelo teórico utilizado en el Mercado relacional [Macchia, Nunzia, 2000, p 17] se pueden fijar las condiciones en cuanto a tecnología ajustándolas al contexto del trabajo en grupo. La primera condición es aprovechar la flexibilidad de la comunicación de doble vía de los medios interactivos. El potencial de Internet: grupos virtuales, correo electrónico, páginas web, etc. La segunda condición consiste en la organización de la información en bases de datos que registren hechos vitales de los procesos requeridos para el grupo, teoría, documentos y todo lo que resulte de la interacción de los miembros. El objetivo central es asegurar la continuidad y la consistencia de la relación. Se agrega una tercera condición, la utilización efectiva de redes informáticas. El trabajo individual en computadoras aisladas es posiblemente menos impactante que el trabajo en computadoras conectadas, los cuales incrementan su funcionalidad ofreciendo acceso a información, recursos y servicios disponibles en computadores remotos ubicados en distintas partes del mundo.

   Según Giga Information Group[5] la tecnología básica de la colaboración incluye cinco grupos:

   a. Mensajería. Tiene penetración en el 98% de las organizaciones.
   b. Repositorios. Bases de Datos de Discusión, documentos, modelos, programas y otras formas de contenido. El repositorio es un conjunto de elementos digitales alrededor del cual se concentran las colaboraciones de los participantes.
   c. Conferencia electrónica. Audioconferencia, videoconferencia, dataconferencia, salas de reunión electrónicas y uso compartido de aplicaciones.

5 Ahora llamado Forrester Research (http://www.forrester.com/) (NASDQ: FORR). Es una compañía independiente de investigación del mercado tecnológico que provee informes acerca del impacto tecnológico en los negocios y las personas.

d. Agendas y programadores. Programación automática de reuniones en función de los roles. Confección de agendas de actividades tomando en cuenta recursos y disponibilidad de tiempo.
e. Automatización de procesos. Flujo de trabajo y otras herramientas de automatización de procesos.

3. **En cuanto a la organización del trabajo**. Plantearse un proyecto piloto previo a la aplicación de la Tecnología del Groupware en la organización completa. Definir la razón de ser del grupo y las razones que inducen su creación definiendo y desarrollando estrategias que apoyen el trabajo colaborativo y nuevas técnicas para compartir proyectos e ideas con grupos externos. Determinar alternativas de acción y la mejor manera de hacer las cosas al establecer: los limites del proyecto de implantación, el Impacto financiero, objetivos, cronogramas de actividades y la determinación de indicadores para medir la productividad del grupo.

4. **En cuanto a la asignación de responsabilidades**. El responsable del grupo tiene entre sus responsabilidades básicas organizar al grupo de trabajo tratando de lograr el mayor consenso posible en cuanto a los puntos a desarrollar y la metodología de trabajo, analizar la información recabada y los resultados obtenidos, coordinar el envío y recepción de mensajes entre los participantes y suministrar respuesta a las peticiones de otros grupos de trabajo relacionados. A su vez, los coordinadores de sub-grupos de trabajo apoyan al responsable principal, agilizan el trabajo del sub-grupo de trabajo, mantienen estrecha comunicación con el responsable acerca de los logros obtenidos y motivan el intercambio de mensajes mediante envío de noticias y recordatorios. Por su parte los participantes o miembros del grupo de trabajo investigan, seleccionan y analizan la información relacionada con la actividad de trabajo del grupo.

2.3 Desarrollo Distribuido de Software

2.3.1. Introducción

Desde hace un tiempo a esta parte las grandes compañías y organizaciones cuentan con la posibilidad de acceder a redes globales. Mas aún, el veloz desarrollo de Internet ha traído consigo un incremento notable en el acceso a tales servicios.

En la actualidad es posible que grupos de desarrolladores situados en distintas partes del mundo puedan trabajar en el desarrollo de un mismo sistema. Desde sus distintas locaciones pueden necesitar modificar miles de archivos diferentes, e inclusive en ocasiones los mismos archivos. Esta manera de trabajar tiene un valor potencialmente alto debido a la creciente necesidad de contar con el personal más calificado disponible de una forma lo más eficiente, cómoda y flexible que sea posible. Al mismo tiempo, esta nueva técnica ha causado cambios considerables en la organización del trabajo en otros aspectos. El hecho de que el staff de desarrollo esté disperso geográficamente afecta el modo en que se realiza la división del trabajo así como el manejo de las interacciones entre los diferentes grupos e individuos. Esto plantea nuevas demandas a las herramientas y sistemas utilizados en el manejo y coordinación del desarrollo, especialmente frente al desarrollo concurrente.

Algunos aspectos del desarrollo de sistemas, tales como la creación de una noción global del proyecto y la comunicación entre desarrolladores y grupos, han permanecido como algo manual y sin un soporte directo de herramientas. Parte de esta necesidad se cubre mediante documentos, especificaciones y reuniones formales. No obstante, puede resultarnos sorprendente que una gran parte de la información para la sincronización de los equipos de desarrollo -ya sea tanto entre los distintos grupos como entre los integrantes de un mismo grupo- en realidad se obtiene a través de contactos informales, por ejemplo durante charlas, reuniones de revisión, intervalos, etc. Cuando la gente que trabaja en temas muy estrechamente ligados está geográficamente dispersa, es necesario evaluar de qué manera es posible contemplar estos aspectos adicionales ya sea en la metodología de trabajo como en la herramienta de soporte.

Una buena estrategia a seguir durante el ciclo del desarrollo consiste en limitar la dependencia entre los desarrolladores, especialmente si están situados en ubicaciones geográficas diferentes. Esto se realiza generalmente al momento de definir la estructura del producto a desarrollar. Así, el sistema es dividido en módulos o componentes, cada uno de cuyos desarrollos se asigna a los diferentes equipos por separado. No obstante esto, suele suceder que a pesar de una adecuada modularización del sistema, persistan algunas dependencias entre las distintas componentes. Estas dependencias se tornan claramente visibles por ejemplo al momento en que las interfaces deban sufrir alguna modificación, o cuando se proceda a la integración de las componentes. En adición a esto, existen cuestiones que si bien no están directamente relacionadas con la creación de módulos,

igualmente requieren ser consideradas durante el desarrollo distribuido, como ser el manejo de la seguridad, el "change management"[6] y la gestión de los repositorios originales del sistema.

2.3.2. Conceptos y motivaciones para el desarrollo distribuido

Definimos al desarrollo distribuido geográficamente como el desarrollo de sistemas de software por parte de un grupo de trabajo cuyos integrantes no están situados en una misma ubicación geográfica. Esto puede significar equipos de desarrollo dispersos en distintos lugares ya sea dentro de una misma ciudad o en distintos países alrededor del mundo.

En un caso extremo, múltiples compañías radicadas en distintos países conforman alianzas temporales, conocidas como corporaciones virtuales, con el propósito de desarrollar un único producto. Al tiempo que tales compañías colaboran juntas en el desarrollo de un producto, puede suceder que además sean competidoras en otros desarrollos.

Este tipo de situaciones representa un serio desafío para el SCM[7], desafío que puede apreciarse en sí mismo desde varios niveles. En un nivel más bajo nos encontramos con la necesidad de distribuir grandes cantidades de datos en tiempo y forma a través de grandes distancias. En el nivel más alto, está la cuestión de cómo integrar el trabajo asincrónico de ingenieros de desarrollo que pueden incluso estar siguiendo diferentes procedimientos y prácticas de configuration management. Todo esto converge en un nivel intermedio, donde reside el problema de proveer adecuada gestión de datos distribuidos acorde a las necesidades de configuration management, en un contexto que ya no puede asumir sino un conjunto descentralizado de entidades que cooperan entre sí.
No obstante esto, cada vez más compañías encuentran sólidas motivaciones de negocios para optar por el desarrollo distribuido. Entre las razones que explican esta tendencia podemos mencionar:

- **La rápida proliferación de PC's** que convierte a la estación de trabajo personal en un elemento crítico en el ambiente de trabajo. Los avances en redes y sistemas abiertos permitieron a los usuarios tomar ventaja de servers especializados sin tener que abandonar sus ambientes locales. Estos servers, a su vez, conducen a la necesidad de un desarrollo distribuido.

- **Cambios radicales de índole organizacional**. Para ser más competitivas, las corporaciones tienden hacia estructuras planas, descentralizadas y de naturaleza global. Asimismo las organizaciones requieren que toda clase de datos relevantes a

6 Change management: se refiere a la administración de cambios de versiones del producto de software que se está desarrollando.
7 SCM: Software Configuration Management. Apunta a la coordinación de los desarrolladores hacia un objetivo común.

la empresa sea accesible a todos los empleados de la misma sin importar dónde se encuentren.

- **Independencia del lugar de trabajo**. El uso de laptops permite copiar archivos de la red y trabajar en forma remota, ya sea en la oficina, en el hogar o cuando se está de viaje.

- **Empleo de estaciones de trabajo personales** que ofrecen mejor relación precio/performance que los grandes mainframes. Los costos más bajos de mantenimiento hacen más sencillo a las compañías mantener estaciones de trabajo en ubicaciones dispersas.

- **Acceso a la información en tiempo real.** Para los empleados brinda la capacidad de tomar mejores y más rápidas decisiones.

- **Escasez y costo de personal de tecnología altamente calificado**. Toda compañía de software y departamento de IT reconoce el desafío de contratar y mantener desarrolladores e ingenieros de software de alta calidad. Así, para poder contar con los mejores "skills" debe pensarse en términos globales. El desarrollo distribuido permite a las grandes compañías "llevar el trabajo al trabajador", permitiendo de este modo beneficiarse con los mejores recursos humanos más allá de toda frontera.

2.3.3. Desafíos del desarrollo distribuido

Desarrollar un único sistema de software mediante equipos de desarrollo distribuidos no es tarea simple. Existen cuestiones de organización, comunicación y de tecnología que deben ser tenidas en consideración para que el proyecto sea exitoso, las cuales se describen a continuación.

2.3.4. Organización

Desde el punto de vista organizativo, deberá determinarse cómo se agrupan los integrantes del proyecto, quién es el líder del grupo, cómo se interrelacionan los distintos grupos, quién es el responsable de tomar decisiones globales al proyecto y quién es el responsable por el éxito del proyecto en general.

En ambientes de desarrollo distribuido es común que existan diferentes culturas y diferentes estilos de programación. Esto se aprecia claramente entre "sites" ubicados en países diferentes.

Estas diferencias culturales introducen obstáculos adicionales –que a veces se manifiestan de manera sutil-, a la consecución del éxito.

Es posible identificar una serie de cuestiones clave que necesitan ser claramente determinadas y comprendidas por todos los integrantes del proyecto, como ser:

- Quién es el responsable del éxito global del proyecto
- Quién es el responsable del gerenciamiento del proyecto
- Quién es el responsable de la arquitectura global del sistema
- Quiénes son los participantes en este proyecto

Los proyectos de desarrollo distribuido tienden a abarcar varias áreas de desarrollo. Es por ello que en general podemos hablar de varios subproyectos que contribuyen a un proyecto global, que llamaremos superproyecto. Todos los miembros de los distintos grupos de desarrollo deberían estar en condiciones de responder las preguntas mencionadas anteriormente, referidas al superproyecto.

Muchos proyectos de desarrollo distribuido suelen fracasar debido a que no establecen la estructura organizacional de este superproyecto. Sin esta infraestructura los equipos tienden a tomar decisiones de arquitectura y tecnologías de forma independiente. A la larga esto hace que el sistema resulte difícil o imposible de integrar. E incluso aún cuando la integración pueda hacerse, la falta de una dirección global puede hacer que los límites entre los subproyectos resulten perceptibles al usuario final.

2.3.5. Comunicación

La comunicación entre los distintos equipos se ve resentida por el hecho de que están situados en puntos geográficos distantes. Incluso hoy en día con el uso de e-mail, fax, voice mail y video conferencias, dos equipos que no están en el mismo sitio tienen muchos menos canales de comunicación que aquellos que sí lo están. El valor del entorno comunitario que establece la gente trabajando en un mismo lugar no puede ser subestimado. Por eso resulta importante maximizar la capacidad de los equipos para intercomunicarse al mismo tiempo que se debe minimizar la cantidad de comunicación necesaria entre ellos para el día a día.

Para alcanzar este objetivo los equipos deben tener una visión común de como funcionará el sistema, expresada en una arquitectura preacordada. Dividir arbitrariamente las funcionalidades a ser desarrolladas entre equipos separados geográficamente raramente conduce al éxito.

Debe establecerse primero una arquitectura del sistema; luego dividirla en componentes desarrollables separadamente y asignar cada componente a un equipo de desarrollo. De esta manera, los canales de comunicación deberán ser fuertes durante la definición de la arquitectura e integración del sistema, pero durante el desarrollo de cada componente los equipos sólo necesitarán comunicarse cuando las interfaces del componente requieran alguna aclaración o modificación.

2.3.6. Tecnología

El desarrollo distribuido involucra dos herramientas tecnológicas: tecnologías de implementación y tecnologías de desarrollo. Tecnologías de implementación son aquellas utilizadas en la creación del sistema que se desarrolla (por ejemplo el sistema operativo, componentes GUI[8], lenguajes de programación y bases de datos). Por otra parte, tecnologías de desarrollo son aquellas utilizadas para el establecimiento de un entorno efectivo de desarrollo distribuido de software (por ejemplo la herramienta SCM, la herramienta de seguimiento de fallos, el compilador, la IDE[9]).

2.3.6.1. Tecnologías de implementación

La tecnología utilizada en el sistema de software debería estar claramente especificada como parte de la arquitectura del mismo. Ciertamente es posible que los diferentes grupos de trabajo utilicen distintas tecnologías para el desarrollo de sus componentes, pero esto puede tener efectos que se pondrán de manifiesto en el producto final, tales como en el uso, facilidad de instalación, facilidad de administración, aspecto, etc. Debería entonces evitarse el uso de tecnologías diferentes, a no ser que existan importantes razones de negocio que lo justifiquen y se sepa de antemano cómo este hecho podrá impactar en el sistema final.

2.3.6.2. Tecnologías de desarrollo

La segunda decisión de índole tecnológica que enfrentan los equipos de desarrollo distribuido es la de determinar cuál será la tecnología utilizada para construir y administrar el desarrollo del sistema de software. Este concepto es comúnmente conocido como "entorno de desarrollo del software". Deberán determinarse las herramientas y procesos a ser utilizados por los equipos integrantes del proyecto, asegurando al mismo tiempo que dichas herramientas puedan soportar el desarrollo distribuido.

Si bien es posible –y en ciertos casos necesarios- recurrir al empleo de distintos ambientes de desarrollo, hay que subrayar que el número potencial de problemas que enfrentarán los equipos de trabajo se verá notablemente reducido si se puede optar por el uso de un solo set de herramientas y procesos.

2.3.7. Casos de desarrollo distribuido

El desarrollo distribuido puede surgir por varias razones. A continuación se clasifican estas razones recurriendo a casos que son característicos, como ser:

- Trabajo a distancia
- Outsourcing
- Grupos distribuidos, integrantes no dispersos
- Grupos distribuidos, integrantes dispersos

8 GUI: Graphic User Interface: Interfaz gráfica del usuario.
9 IDE: Integrated Development Environment. Entorno integrado de desarrollo.

Los distintos casos planteados pueden ocurrir individualmente o en combinación. Así por ejemplo pueden existir varios grupos de trabajo que normalmente se encuentran trabajando todos juntos pero que ocasionalmente pueden estar distribuidos geográficamente.

2.3.8. Trabajo a distancia

Entendemos por trabajo a distancia al trabajo de breve duración que se realiza en cualquier otro sitio fuera del lugar usual de trabajo. Como ejemplo clásico puede citarse el caso de trabajar en el propio hogar como un complemento del trabajo diario. Cuando el desarrollador trabaja en su casa (o en otro sitio) ya con carácter habitual o por períodos extensos, nos encontramos en una situación que se asemeja al de un grupo distribuido, como se verá más adelante.

Esta manera de trabajar se caracteriza por contar con una capacidad de computación de posibilidades limitadas (por ejemplo una laptop) y un medio de comunicación con el mundo externo relativamente lento (por ejemplo a través de un módem de datos conectado al lugar habitual de trabajo). No obstante estas limitaciones, la preparación del ambiente de trabajo debe poder hacerse en forma rápida ya que el tiempo que dura el trabajo en cada sesión es relativamente corto (por ejemplo unas pocas horas a la noche).

Debido a que se siguen manteniendo los contactos habituales del día a día, la posibilidad de la comunicación informal y el mantenimiento del espíritu de equipo son más o menos iguales que en el caso del desarrollo local.

Dos metodologías comunes de trabajo son:

- Llevarse archivos individuales al propio hogar y trabajar con ellos de modo local. Esta manera de trabajar es necesaria cuando no es viable tener en casa el ambiente completo de desarrollo o bien porque resultaría muy costoso en términos de tiempo el tener que realizar actualizaciones permanentemente.
- Conectarse en forma remota al lugar de trabajo, de modo que la computadora personal pasa a ser una terminal más.

2.3.9. Outsourcing

Otra alternativa al desarrollo propio o a la compra de componentes existentes (COTS – Commercial Off The Shelf) consiste en contratar a terceros que se ocupen del desarrollo de uno o varios componentes. Esta manera de trabajar es conocida como outsourcing (o subcontratación) y brinda, comparada con el uso de COTS, un mayor control sobre el desarrollo del componente, si bien a un costo más elevado. El outsourcing se basa en una estrecha colaboración entre proveedor y cliente. Consecuentemente, el cliente a menudo se encarga de proveer el ambiente de prueba, de modo que el proveedor tenga la posibilidad

de testear el componente en un entorno similar al del ambiente final antes de proceder a su entrega.

Dado que el contratante es el responsable final del producto, posibles cuestiones de error/change management pueden verse reflejadas en solicitudes de cambio en el componente, dirigidas hacia su proveedor. Como en el caso cualquier solicitud, debe quedar claro qué es lo que el proveedor debería entregar, pero adicionalmente habrá que tener en cuenta que el ambiente del componente puede haber sufrido cambios.

Desde la perspectiva de change management:

- El contratista deberá ser capaz de integrar eventuales nuevas versiones del componente en el producto, el cual a su vez puede haberse modificado desde la última versión del componente.
- El proveedor deberá ser capaz de manejar las actualizaciones y ambientes de prueba.
- El contratista y el proveedor no necesariamente cuentan con las mismas herramientas de SCM, lo cual puede dificultar la actualización (en ambas direcciones).
- Al momento de la entrega de los programas fuentes, las herramientas de generación del código deberán ser consistentes entre el proveedor y el contratista.

2.3.10. Grupos distribuidos, conexos (integrantes no dispersos)

Los desarrolladores que trabajan en diferentes compañías afiliadas generalmente están asignados a grupos o proyectos de carácter local. En este contexto la división del trabajo ya queda determinada al momento de la estructuración del proyecto global, de manera de evitar una gran dependencia entre los distintos grupos.

El producto a desarrollar se divide en subproductos, cuyos desarrollos pueden ser asignados a distintos grupos de trabajo. Esta división posibilita que la mayor parte del desarrollo sea hecho a nivel local dentro de dichos grupos, minimizando la necesidad de comunicación entre los mismos. Dentro de cada grupo, la situación se asemeja al caso del desarrollo local. Los grupos situados en lugares distintos normalmente sólo tienen acceso a las últimas –y más estables- versiones producidas por los otros grupos. Debido a las distancias geográficas, los problemas que pudieran surgir serán potencialmente más difíciles de resolver, por lo cual la actualización y distribución entre los distintos grupos requiere de mayor control y esfuerzo. Estas actividades pueden considerarse entregas internas. La cooperación entre los distintos grupos puede verse facilitada si el trabajo se planifica en fases, de las cuales cada uno de ellos está concientizado. Como contrapartida, este enfoque presenta la desventaja de que una eventual redistribución o nueva división del trabajo resulta ser más dificultosa.

Desde la perspectiva de configuration management (CM)[10]:

- Los archivos residen en distintos file systems pero –idealmente- en el mismo sistema de configuration management. Sin embargo, las grandes corporaciones pueden tener distintos sistemas de CM en sus compañías afiliadas.
- Cuando las ubicaciones son permanentes, todos los grupos deberían ser capaces de trabajar dentro de un completo entorno de desarrollo y contar con la posibilidad de realizar pruebas.
- Los grupos se entregan subproductos entre ellos, en lugar de desarrollar en conjunto.
- Usualmente existen pocos o ningún contacto no planificado entre los distintos grupos. La comunicación se limita, por ejemplo, a reuniones semanales que pueden ser en persona o a través de conferencias telefónicas, videoconferencias, etc.
- Es importante no perder de vista el status del desarrollo entre los distintos grupos.
- El change management de componentes comunes tales como interfaces, adquiere especial importancia.

2.3.11. Grupos distribuidos, disconexos (integrantes dispersos)

El escenario de grupos de trabajo distribuidos con miembros dispersos implica que personas trabajando en el mismo proyecto, quizás incluso con los mismos archivos, están geográficamente separadas. Así, la posibilidad del contacto diario habitual -ya sea formal o informal- se pierde incluso dentro de un mismo grupo de trabajo.

Los distintos proyectos que intervienen en el desarrollo de un producto usualmente cuentan con ciertos componentes y librerías comunes. Sin bien los cambios que pudieran sufrir las mismas son contados -simplemente porque son de uso común y las modificaciones no son fácilmente manejables- a veces son inevitables. Si miembros de un grupo ubicados en sitios distintos quieren hacer modificaciones simultáneamente en archivos comunes, nos encontramos con una situación similar a la de modificación de interfaces por parte de grupos distribuidos conexos (es decir sin integrantes dispersos) sólo que en este caso el problema aplica a todos los archivos.

Hay que tener en claro que individuos dispersos se pueden agrupar en pequeños grupos conexos. No obstante estos esfuerzos, existen situaciones en las que los grupos necesitan trabajar más estrechamente aunque sus integrantes estén separados. Un ejemplo obvio es cuando algunas personas pertenecientes a un mismo grupo deben viajar para reunirse con otro grupo. Desde ya existe el deseo de continuar trabajando con el grupo habitual, lo cual se puede lograr como grupo distribuido. Otro caso similar surge cuando ciertos integrantes del staff se mudan a un proyecto nuevo pero precisan ser consultados regularmente para el

10 Configuration management: Administración de configuración. Corresponde al registro detallado y actualizado de información que describe a un sistema de computación empresarial.

proyecto anterior. Mucha gente con “skills” especiales suele ser incluida en varios grupos, los cuales pueden estar ubicados en distintas locaciones.

Desde la perspectiva de configuration management:

- Es importante que los integrantes del grupo reciban información acerca de lo que el resto está haciendo, cómo se está desarrollando el proyecto, su status, qué cambios se llevaron a cabo y por quién, etc.
- Se deberá soportar la modificación simultánea de archivos
- Las soluciones que recurren a “locks” y accesos exclusivos a los archivos no suelen ser adecuadas ya que resulta dificultoso resolver las situaciones que surgen cuando los integrantes ubicados en lugares diferentes deben quedar esperándose unos a otros

Cabe destacar que si se considera que el desarrollo de software será derivado geográficamente a otro país, existe una modalidad de trabajo últimamente difundida que se llama Offshore.

2.3.12. ¿Qué es el Desarrollo Offshore?

El Desarrollo Offshore es la Subcontratación de servicios de desarrollo de Software a una compañía ubicada en un país extranjero, es decir que se da cuando una empresa obtiene un Sistema que necesita para usarlo en su país, cuyo desarrollo fue efectuado por una compañía extranjera. De manera que toda la comunicación entre ambas organizaciones y la entrega del producto resultante se realiza mediante Internet.

El desarrollo Offshore ha surgido como una alternativa viable para la contratación de desarrollos de software, no solo por la decisiva ventaja a nivel costos, sino por la calidad de los productos finales. Este modelo provee acceso a soluciones de categoría mundial en países donde los costos operativos son mucho más bajos.

Por una parte, se consigue una sustancial reducción de costos, ya que el software es desarrollado en países con un costo de desarrollo menor que al de aquellos en los que va a ser usado. Y por otra parte, se consigue una mejor gestión de la carga laboral, así cuando una empresa tiene “picos” de trabajo en el área de desarrollo, puede resolverlos fácilmente sin tener que contratar a nuevos desarrolladores, los cuales son un gasto innecesario en épocas con baja carga laboral.

Ventajas:

- Competencia técnica: Los mejores profesionales del área de IT están disponibles y pueden trabajar para usted.
- Calidad: los proyectos se realizarán en los tiempos fijados, y de acuerdo a la más alta calidad y estándares técnicos.

- Costos: los clientes ahorrarán al menos un 35 por ciento de todos sus gastos comparados al desarrollo realizado "in-house", sin tener en cuenta los gastos de infraestructura y equipamiento.
- Protección: Todo derecho de propiedad intelectual le pertenecerá a la compañía original.

Inconvenientes:

- Offshore no es el sistema de trabajo más adecuado para todas las compañías.
- La empresa contratista ha de seleccionar a la empresa de offshore adecuada y ésta no es tarea fácil.
- La empresa contratista ha de dedicar recursos propios para el establecimiento de los requerimientos, gestión y seguimiento del proyecto.
- Problemas de comunicación entre las partes. La comunicación es clave, y se han de establecer líneas claras de comunicación y mantenerse viva a lo largo de la vida del proyecto. El mayor porcentaje de fracasos de proyectos offshore se debe a una comunicación inadecuada.

2.4 Metodología de desarrollo de software

En primer lugar, deseamos clarificar qué entendemos por metodología, el diccionario de la Real Academia Española define a la metodología como "la ciencia del método" y a método como el "modo de decir o hacer con orden una cosa" y también como el "modo de obrar o proceder".

Podemos utilizar dicho término dándole las siguientes acepciones:
Método: conjunto de etapas que se llevan a cabo en un orden determinado y que mantienen entre sí una estrecha relación con el propósito de estudiar un sistema de información.
Metodología: estudio del método que se utiliza en el análisis, diseño e implementación de sistemas de información.

¿Cuál es la justificación para desarrollar y hacer conocer una metodología?
Hay varias razones. En primer lugar pensamos que no existe ningún misterio asociado con la tarea de sistemas y que, por el contrario, es beneficioso que tanto usuarios como hombres de sistemas tengan una muy clara idea de cuáles son los propósitos de cada etapa y sus respectivas fases (sub-etapas). Otra razón fundamental es que únicamente a través de una metodología ordenada y clara es posible obtener resultados satisfactorios de un trabajo en equipo, pues ella permite coordinar y hacer comprender a cada integrante del grupo de trabajo su participación e integración de su respectiva tarea con las del resto.

Por lo tanto podemos decir que las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos software.

En definitiva una metodología va indicando paso a paso todas las actividades a realizar para lograr el producto informático deseado, indicando además qué personas deben participar en el desarrollo de las actividades y qué papel deben de tener. Además detallan la información que se debe producir como resultado de una actividad y la información necesaria para comenzarla.

Una metodología de desarrollo de software permite producir organizada y económicamente software de alta calidad, siguiendo una serie de pasos donde se utilizan un conjunto de técnicas, notación y normas de documentación preestablecidas.

El análisis y diseño, como elementos esenciales del proceso de desarrollo, obligan a tener especial atención y por tal motivo se han ido creando metodologías que sirven de base para tomar las decisiones que afectarán el producto final. Con el advenimiento de la disciplina de la ingeniería del software se inicia el proceso de desarrollo de metodologías, las primeras de ellas fueron las estructuradas, y en forma posterior aparecen las metodologías orientadas a objetos, siendo estas últimas las más difundidas actualmente en el medio.

Una metodología de desarrollo de software presenta una forma de modelar el mundo real con el fin de llevarlo al dominio del computador, a través del modelo se puede obtener una visión global del sistema, para facilitar la especificación de los requerimientos, las restricciones del sistema y la solución del problema.

Las técnicas indican cómo debe ser realizada una actividad técnica determinada identificada en la metodología. Combina el empleo de unos modelos o representaciones gráficas junto con el empleo de unos procedimientos detallados. Se debe tener en consideración que una técnica determinada puede ser utilizada en una o más actividades de la metodología de desarrollo de software. Además se debe tener mucho cuidado cuando se quiere cambiar una técnica por otra.

2.4.1. Metodologías existentes para el desarrollo de software

Muchas veces las compañías no encuentran la metodología más adecuada para su estilo organizacional y terminan por hacer o diseñar una metodología propia, algo que por supuesto no está mal, siempre y cuando cumpla con el objetivo. Una metodología consiste en concretar el tipo de ciclo de vida que se va a seguir y la forma en la que se realizan las actividades dentro de cada etapa. Si analizamos detenidamente el ciclo de vida del desarrollo del producto, cada etapa tiene datos de entrada y resultados específicos, las

cuáles presentan riesgos que son importantes atacar a tiempo. Es por eso la importancia del estudio de los diferentes ciclos de vida.

No existe una metodología universal para hacer frente con éxito a cualquier proyecto de desarrollo de software. Toda metodología debe ser adaptada al contexto del proyecto: recursos técnicos y humanos, tiempo de desarrollo, tipo de sistema, etc.
Históricamente, las metodologías tradicionales en el campo de desarrollo de software, han intentado abordar la mayor cantidad de situaciones de contexto del proyecto, exigiendo un esfuerzo considerable para ser adaptadas, sobre todo en proyectos pequeños y con requisitos muy cambiantes. En su lugar las metodologías ágiles emergen como una posible respuesta para llenar ese vacío metodológico. Por estar especialmente orientadas a proyectos pequeños y a la gestión de riesgos, las metodologías ágiles constituyen una solución a la medida para ese entorno.

2.4.2. Etapas básicas del ciclo de vida del desarrollo de software

Al igual que en otros sistemas de ingeniería, los sistemas de software requieren un tiempo y esfuerzo considerable para su desarrollo y deben permanecer en uso por un período mucho mayor. Durante este tiempo de desarrollo y uso, desde que se detecta la necesidad de construir un sistema de software hasta que este es retirado, se identifican varias etapas que en conjunto se denominan el ciclo de vida del software y en cada caso, en función de cuáles sean las características del proyecto, se configurará el ciclo de vida de forma diferente. Las etapas principales a realizar en cualquier ciclo de vida son:

1. *Análisis de requerimientos*: El objetivo de esta fase es identificar los requerimientos del sistema.

2. *Diseño*: El objetivo de esta fase es desarrollar una representación coherente y organizada del producto "software" que satisfaga la especificación de requerimientos. La estructura de datos (diseño de datos), la arquitectura del software (diseño arquitectónico), la caracterización de las interfaces (diseño de interfaces) y la interfaz de usuario, son realizadas en esta fase.

3. *Codificación*: En esta fase se lleva a cabo la traducción del diseño a un lenguaje de programación que pueda ser luego interpretado por la máquina. Se pretende traducir el diseño en un código fuente. El código fuente debe estar acompañado de la documentación correspondiente, que constituye la manifestación física del diseño de acuerdo con los estándares y metodologías adoptados para el proyecto.

4. *Pruebas*: Esta fase constituye la revisión final de las especificaciones, el diseño y la codificación y puede ser considerada crítica para asegurar la calidad del producto generado. En ella se ejecutará el software con determinados datos de entrada, observando los resultados que se producen y comparándolos con los que,

teóricamente y según las especificaciones, el sistema debería producir, detectando así posibles fallos.

5. *Mantenimiento*: En esta fase, que tiene lugar después de la entrega se asegura que el sistema siga funcionando y adaptándose a nuevos requisitos.

2.4.3. Métodos tradicionales o formales de desarrollo de software

Las formas de organizar y estructurar la secuencia de ejecución de las tareas en las diferentes fases de cada uno de los métodos pueden dar lugar a un tipo de ciclo de vida diferente. A continuación se describen brevemente los modelos de ciclos de vida de los métodos tradicionales y los métodos alternativos (ágiles) que se utilizan actualmente en compañías pequeñas de desarrollo de producto en plazos de tiempo cortos.

Uno de los grandes pasos dados en la industria del software lo fue el desarrollo del modelo en Cascada. El mismo surge como respuesta al modelo Codificar y Probar que era el que predominaba en la década de los ‘60. A continuación se describe el modelo en Cascada.

2.4.4. Método de ciclo de vida clásico (Modelo en Cascada)

Este método fue aplicado con éxito para estructurar y gestionar grandes proyectos de software en importantes compañías de desarrollo. Este modelo [Rancan, 2003] puede ser visto como un modelo con forma de Cascada de agua de varios saltos, en la que cada salto representa cada una de las fases del ciclo de vida. La evolución del producto software se puede ver como una secuencia ordenada de transiciones, de fase en fase, que evoluciona en forma lineal, exige un enfoque sistemático y secuencial del desarrollo del producto software y contempla las siguientes fases: ingeniería y análisis del sistema global, análisis de requisitos de software, diseño, codificación, prueba y mantenimiento.

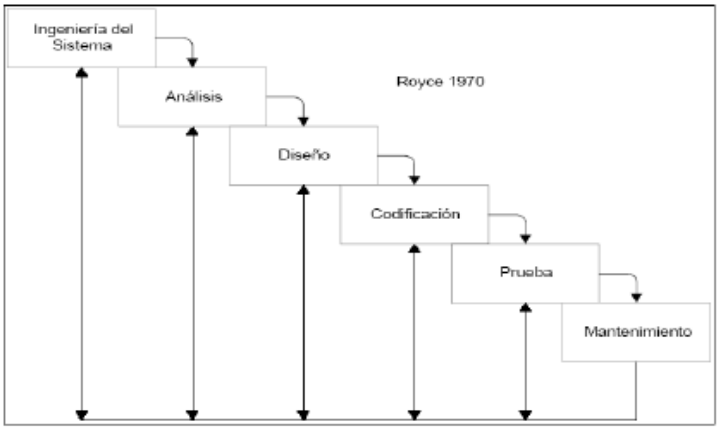

Figura 1. Ciclo de vida básico [Rancan, 2003]

De esta manera y en forma bastante temprana en algunos casos, fueron surgiendo diversos procesos denominados iterativos que proponían lidiar con la poca predictibilidad del

software (mejorando muchas de las fallas del modelo en Cascada), mitigando de esta forma los riesgos en forma temprana. Posterior a los procesos iterativos, se comenzaron a desprender diversas ramas; como son el modelo Iterativo e Incremental, el modelo en Espiral, el modelo basado en Prototipo, entre otros. Básicamente, la postura de estos modelos es la de basar el desarrollo en iteraciones e ir construyendo la aplicación en forma progresiva, agregando funcionalidad sucesivamente. Las iteraciones representan un mini-proyecto, el cual está compuesto por todas las fases del desarrollo (requerimientos, diseño, implementación y prueba. A continuación se describen algunos de estos modelos.

2.4.5. Modelo en Espiral Win Win

Boehm [1988] fue el autor que desarrollo el modelo espiral. De este surgió una de las ideas fundamentales que las metodologías posteriores adoptarían: el análisis de riesgos. Este modelo de carácter iterativo en sus primeras fases, plantea la necesidad de realizar al principio diversas iteraciones dirigidas a mitigar los riesgos más críticos encontrados en el proyecto, mediante la realización de prototipos o simulaciones desechables tendientes a probar algún concepto. Una vez que esos prototipos son validados, se suceden iteraciones del tipo de determinar objetivos, evaluar, desarrollar y planear.

Es ideal para manejar proyectos que requieran la incorporación de nuevas tecnologías, o para desarrollar productos completamente nuevos o con un nivel alto de inestabilidad de los requerimientos. Esta nueva filosofía se encuentra representada por ciclos de desarrollo evolutivo e iterativo en forma de espiral, cuyo avance angular representa el progreso del desarrollo, en tanto que el desplazamiento radial desde el centro hacia fuera indica el incremento de los costos de desarrollo en forma acumulativa.

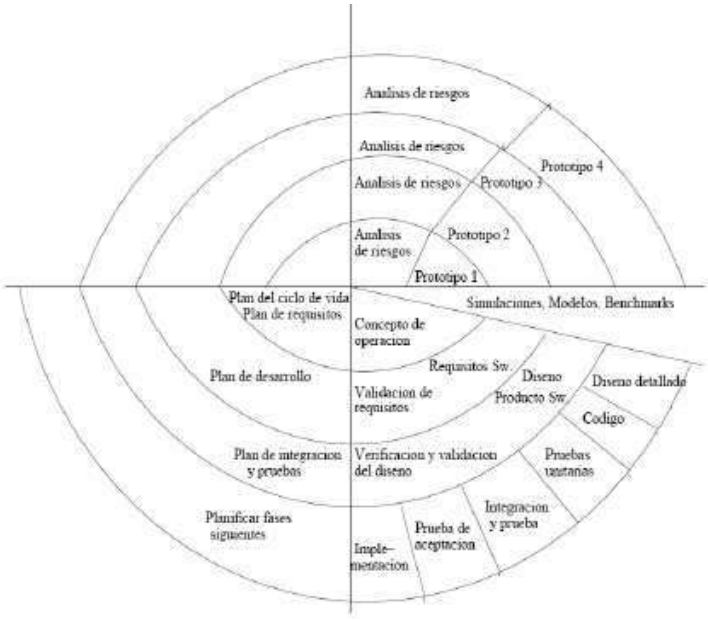

Figura 2. Modelo Espiral [Álvarez, 2002]

2.4.6. Métodos ágiles e iterativos para desarrollo de software

A principios de la década del ’90, surgió un enfoque que fue bastante revolucionario para su momento ya que iba en contra de la creencia de que mediante procesos altamente definidos se iba a lograr obtener software en tiempo, costo y con la calidad requerida. El enfoque fue planteado por primera vez por Martin Fowler [1991] y se dió a conocer en la comunidad de ingeniería de software con el mismo nombre que su libro, 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. Estos modelos se basan en iteraciones. El desarrollo iterativo es un método de construcción de productos cuyo ciclo de vida está compuesto por un conjunto de iteraciones, las cuales tienen como objetivo entregar versiones del software.

En febrero de 2001, tras una reunión celebrada en Utah, EEUU, nace el término Ágil aplicado al desarrollo de software. En esta reunión participan un grupo de 17 expertos de la industria del software, incluyendo algunos de los creadores o impulsores de metodologías de software. Su objetivo fue 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 as actividades desarrolladas.
Tras esta reunión se creó *The Agile Alliance*[11], 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 término “Métodos Ágiles” [Larman, 2003], se utiliza para definir aquellos métodos que estaban surgiendo como alternativa a las metodologías formales, las cuáles se consideran excesivamente “pesadas” y rígidas por su carácter normativo y fuerte dependencia de planificaciones detalladas, previas al desarrollo. Las metodologías ágiles reconocen las características inherentes de complejidad del software, y el carácter empírico que debe tener un proceso de desarrollo del mismo. Entre las metodologías ágiles más destacadas hasta el momento podemos nombrar:

- XP – Extreme Programming
- Scrum
- Crystal Clear
- DSDM – Dynamic Systems Developmemt Method
- FDD – Feature Driven Development
- ASD – Adaptive Software Development
- XBreed

11 www.agilealliance.com

- Extreme Modeling

Estos métodos se guían por cuatro postulados que comprende el espíritu en el que se basan estos métodos; lo cual se denomina "Manifiesto Ágil". A continuación se describe los postulados y principios ágiles:

2.4.6.1. Postulados del Manifiesto Ágil

Estos métodos valoran:

- Los individuos y su interacción, por encima de los procesos y las herramientas.
- El desarrollo del software, por encima de la documentación exhaustiva.
- La colaboración con el cliente, por encima la negociación contractual.
- La respuesta al cambio, por encima del seguimiento de un plan.

2.4.6.2. Principios del Manifiesto Ágil

Los 12 principios del Manifiesto Ágil plantean principios que pueden resultar viables para los proyectos de software de determinadas características. A continuación se describe cada uno de ellos:

- Nuestra principal prioridad es satisfacer al cliente a través de la entrega temprana y continua de software de valor.
- Son bienvenidos los requisitos cambiantes, incluso si llegan tarde al desarrollo. Los procesos ágiles se doblegan al cambio como ventaja competitiva para el cliente.
- Entregar frecuentemente software, en períodos de un par de semanas hasta un par de meses, con preferencia en períodos breves.
- Las personas del negocio y los desarrolladores deben trabajar juntos de forma cotidiana a través del proyecto.
- Construcción de proyectos en torno a individuos motivados, dándoles la oportunidad y el respaldo que necesitan y procurándoles confianza para que realicen la tarea.
- La forma más eficiente y efectiva de comunicar información de ida y vuelta dentro de un equipo de desarrollo es mediante la conversación cara a cara.
- El software desarrollado es la principal medida del progreso.
- Los procesos ágiles promueven el desarrollo sostenido. Los patrocinadores, desarrolladores y usuarios deben mantener un ritmo constante de forma indefinida.
- La atención continua a la excelencia técnica enaltece la agilidad.
- La simplicidad como arte de maximizar la cantidad de trabajo que se hace, es esencial.
- Las mejores arquitecturas, requisitos y diseños emergen de equipos que se auto-organizan.
- En intervalos regulares, el equipo reflexiona sobre la forma de ser más efectivo y ajusta su conducta como consecuencia.

A continuación, se describen las metodologías más importantes:

2.4.7. Extreme Programming (XP)[12]

Es una de las metodologías de desarrollo de software más exitosas en la actualidad, utilizadas para proyectos de corto plazo. Su creador, Kent Beck[13] fue el padre del Manifiesto Ágil. Su principal asunción es que con un poco de planificación, un poco de codificación y unas pocas pruebas se puede decidir si se está siguiendo un camino acertado o equivocado, evitando así tener que echar marcha atrás demasiado tarde. La metodología 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 al éxito del proyecto. XP se basa en retrocomunicación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define especialmente para proyectos con requerimientos imprecisos y muy cambiantes, y para proyectos donde exista un alto riesgo técnico [Grupo ISSI, 2003].

2.4.8. Scrum[14]

Uno de los análisis más importantes en está metodología desembocó en un libro escrito por dos de sus creadores, Ken Schwaber y Mike Beedle. [Schwaber, 2001]. Scrum define métodos de gestión y control para complementar la aplicación de otros métodos ágiles como XP, que centrados en prácticas de tipo técnico, carecen de ellas. Los principios de Scrum son:

- Equipos autogestionados.
- Una vez dimensionadas las tareas no es posible agregarles trabajo extra.
- Reuniones diarias con los miembros del equipo.
- Iteraciones de desarrollo de frecuencia inferior a un mes, al final de las cuales se presenta el resultado a los externos del equipo de desarrollo, y se realiza una planificación de la siguiente iteración, guiada por cliente.

Scrum tiene 3 elementos básicos que son: el “Sprint”, el “Product Backlog”, el “Sprint Backlog”.

El “Product Backlog” es una lista con las funcionalidades que debe tener el software que se está desarrollando. Esta lista debe estar ordenada por orden de importancia de las funcionalidades comenzando por la más importante. En esta lista no es necesario que figuren todas las funcionalidades, pero si se deben listar las mas importantes.

De las funcionalidades listadas en el Product Backlog, se toman las más importantes y se descomponen en una serie de tareas con las cuales se confecciona otra lista denominada

12 ww.extremeprogramming.org,www.xprogramming.com,c2.com/cgi/wiki?ExtremeProgramming
13 Kent Beck: es uno de los creadores de la metodología ágil para el desarrollo de software conocida como programación extrema (XP).
14 www.controlchaos.com

"Sprint Backlog" y que representa las tareas que se deberán ejecutar en el próximo período, al cual se lo denomina "Sprint". La duración de los sprints no siempre es la misma e incluso puede ser diferente de acuerdo al momento del proyecto en el que se esté.
La idea de Scrum es la de dividir el proyecto en varios sprints en cada uno de los cuales se obtendrá como resultado una nueva versión del software que cumplirá con las funcionalidades del Product Backlog abarcadas por el Sprint.
Según lo planteado por esta técnica, se propone tres etapas en las cuales tanto los desarrolladores como los usuarios juegan un papel muy importante.

El proceso de Scrum se puede dividir en tres etapas (ver Figura 3):

Pre-Juego: En esta etapa se definen las funcionalidades que formarán parte del Product Backlog y a partir de éste se define el Sprint Backlog que se utilizará en el Sprint. Además, se define la arquitectura se realiza el diseño de alto nivel.

Juego: La etapa de juego es aquella durante la cual se desarrollan las tareas definidas en el Sprint Backlog. Para poder realizar un seguimiento de la marcha de las actividades a realizar en esta etapa se realizan reuniones diarias entre los miembros del equipo, denominadas "Stand Up Meetings", donde cada miembro debe responder las siguientes tres preguntas:
¿Qué hice ayer?
¿Qué voy a hacer hoy?
¿Qué ayuda necesito?

Post Juego: Finalmente, en esta etapa se evalúan los resultados del Sprint una vez finalizado el mismo. Aquí no sólo se verifica que se hayan cumplido con las funcionalidades predefinidas para el Sprint sino que también se evalúan los tiempos, el progreso del proyecto y de ser necesario se redefinen algunos tiempos del proyecto. Si se trata de una iteración dentro del proyecto, se vuelve a la Etapa 1 para planificar el próximo Sprint, en cambio si es la última iteración se procederá a confeccionar toda la documentación necesaria para la finalización del mismo.

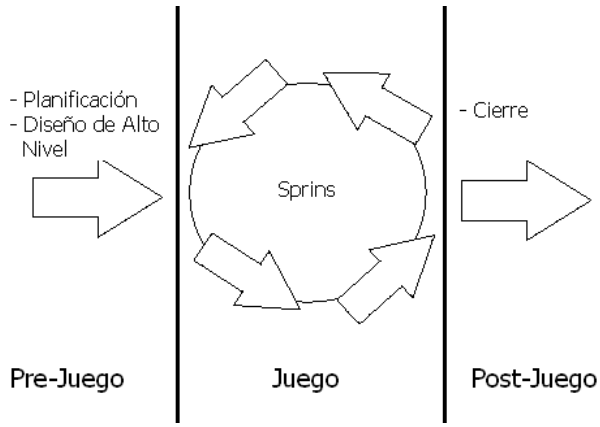

Figura 3. Interacción entre las capas de Scrum [Scrum, 1995]

2.4.9. Crystal Methodologies[15]

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[16] producidos. Han sido desarrolladas por Alistair Cockburn[17]. 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).

A continuación se describen los modelos de propiedad comercial:

2.4.10. Microsoft Solution Framework (MSF)

MSF es la metodología empleada por Microsoft para el desarrollo de software. Esta es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas [Sánchez, 2004]. El marco MSF se asienta sobre unos principios fundamentales que definen la cultura del entorno de desarrollo:

1. Fomentar la comunicación abierta.
2. Trabajar en torno a una visión compartida.
3. Motivar a los integrantes del equipo.
4. Establecer responsabilidades claras y compartidas.
5. Centrar el objetivo en la entrega de valor para el negocio.
6. Permanecer ágiles y esperar al cambio.
7. Invertir en calidad.
8. Aprender de la experiencia.

2.4.11. Método RUP (Proceso Unificado)

RUP es un "modelo-producto" desarrollado y mantenido por Rational Software, integrado en su conjunto de herramientas de desarrollo, y distribuido por IBM. La metodología se conoce como RUP, llamada así por sus siglas en inglés Rational Unified Process.

El proceso unificado consiste en una serie de ciclos. Al final de cada ciclo se tiene una versión del producto. Los objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes Las fases de cada ciclo en este método: inicio, elaboración, construcción y transición. [RUP, 2002].

15 www.crystalmethodologies.org

16 Artefacto: término utilizado en el proceso de desarrollo de software denominado RUP (en inglés, Rational Unified Process). Se refiere a los elementos de información producidos, modificados o usados por el proceso.

17 Alistair Cockburn: reconocido experto internacional en tecnología de orientación a objetos y gestor de proyectos, es consultor deCockburn Associates.

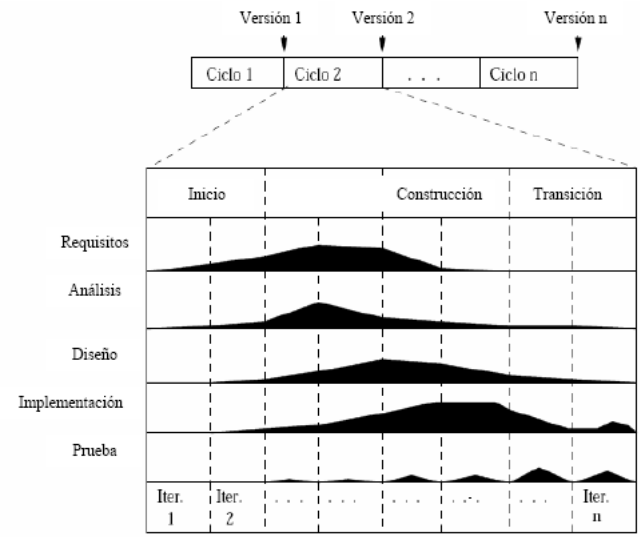

Figura 4. Ciclo de vida del proceso unificado [RUP, 2002]

2.4.12. ¿Qué hay que saber para construir o elegir una metodología?

En el momento de adoptar un estándar o construir una metodología, se han de considerar unos requisitos deseables, por lo que seguidamente se proponen una serie de criterios de evaluación de dichos requisitos.

La metodología debe ajustarse a los objetivos

Cada aproximación al desarrollo de software está basada en unos objetivos. Por ello la metodología que se elija debe recoger el aspecto filosófico de la aproximación deseada, es decir que los objetivos generales del desarrollo deben estar implementados en la metodología de desarrollo.

La metodología debe cubrir el ciclo entero de desarrollo de software

Para ello la metodología ha de realizar unas etapas:

- Investigación
- Análisis de requisitos
- Diseño

La metodología debe integrar las distintas fases del ciclo de desarrollo

Trazabilidad. Es importante poder referirse a otras fases de un proyecto y fusionarlo con las fases previas. Es importante poder moverse no sólo hacia adelante en el ciclo de vida, sino hacia atrás de forma que se pueda comprobar el trabajo realizado y se puedan efectuar correcciones.

Fácil interacción entre etapas del ciclo de desarrollo. Es necesaria una validación formal de cada fase antes de pasar a la siguiente. La información que se pierde en una fase determinada queda perdida para siempre, con un impacto en el sistema resultante.

La metodología debe incluir la realización de validaciones
La metodología debe detectar y corregir los errores cuanto antes. Uno de los problemas más frecuentes y costosos es el aplazamiento de la detección y corrección de problemas en las etapas finales del proyecto. Cuanto más tarde sea detectado el error más caro será corregirlo.

Por lo tanto cada fase del proceso de desarrollo de software deberá incluir una actividad de validación explícita.

La metodología debe soportar la determinación de la exactitud del sistema a través del ciclo de desarrollo.
La exactitud del sistema implica muchos asuntos, incluyendo la correspondencia entre el sistema y sus especificaciones, así como que el sistema cumple con las necesidades del usuario. Por ejemplo, los métodos usados para análisis y especificación del sistema deberían colaborar a terminar con el problema del entendimiento entre los informáticos, los usuarios y otras partes implicadas.

Esto implica una comunicación entre usuario y técnico amigable y sencilla, exenta de consideraciones técnicas.

La metodología debe ser la base de una comunicación efectiva.
Debe ser posible gestionar a los informáticos, y éstos deben ser capaces de trabajar conjuntamente. Ha de haber una comunicación efectiva entre analistas, programadores, usuarios y gestores, con pasos bien definidos para realizar progresos visibles durante la actividad del desarrollo.

La metodología debe funcionar en un entorno dinámico orientado al usuario
A lo largo de todo el ciclo de vida del desarrollo se debe producir una transferencia de conocimientos hacia el usuario. La clave del éxito es que todas las partes implicadas han de intercambiar información libremente. La participación del usuario es de importancia vital debido a que sus necesidades evolucionan constantemente. Por otra parte la adquisición de conocimientos del usuario la permitirá la toma de decisiones correctas.

Para involucrar al usuario en el análisis, diseño y administración de datos, es aconsejable el empleo de técnicas estructuradas lo más sencillas posible. Para esto, es esencial contar una buena técnica de diagramación.

La metodología debe especificar claramente los responsables de resultados
Debe especificar claramente quienes son los participantes de cada tarea a desarrollar, debe detallar de una manera clara los resultados de los que serán responsables.

La metodología debe poder emplearse en un entorno amplio de proyectos software
Variedad. Una empresa deberá adoptar una metodología que sea útil para un gran número de sistemas que vaya a construir. Por esta razón no es práctico adoptar varias metodologías en una misma empresa.

Tamaño, vida. Las metodologías deberán ser capaces de abordar sistemas de distintos tamaños y rangos de vida.

Complejidad. La metodología debe servir para sistemas de distinta complejidad, es decir puede abarcar un departamento, varios de departamentos o varias empresas.

Entorno. La metodología debe servir con independencia de la tecnología disponible en la empresa.

La metodología se debe de poder enseñar
Incluso en una organización sencilla, serán muchas las personas que la van a utilizar, incluso los que se incorporen posteriormente a la empresa. Cada persona debe entender las técnicas específicas de la metodología, los procedimientos organizativos y de gestión que la hacen efectiva, las herramientas automatizadas que soportan la metodología y las motivaciones que subyacen en ella.

La metodología debe estar soportada por herramientas CASE [18]
La metodología debe estar soportada por herramientas automatizadas que mejoren la productividad, tanto del ingeniero de software en particular, como la del desarrollo en general.

El uso de estas herramientas reduce el número de personas requeridas y la sobrecarga de comunicación, además de ayudar a producir especificaciones y diseños con menos errores, más fáciles de probar, modificar y usar.

La metodología debe soportar la eventual evolución del sistema
Normalmente durante su tiempo de vida los sistemas tienen muchas versiones, pudiendo durar incluso más de 10 años. Existen herramientas CASE para la gestión de la configuración y otras denominadas "Ingeniería inversa" para ayudar en el mantenimiento

18 CASE (Computer Aided Software Engineering): es el uso de herramientas de software para asistir en el desarrollo y mantenimiento de software. Herramientas utilizadas para asistir de esta manera son conocidas como CASE Tools.

de los sistemas no estructurados, permitiendo estructurar los componentes de éstos facilitando así su mantenimiento.

La metodología debe contener actividades conducentes a mejorar el proceso de desarrollo de software.

Para mejorar el proceso es básico disponer de datos numéricos que evidencian la efectividad de la aplicación del proceso con respecto a cualquier producto software resultante del proceso. Para disponer de estos datos, la metodología debe contener un conjunto de mediciones de proceso para identificar la calidad y coste asociado a cada etapa del proceso.

2.5 Gestión de Proyectos

2.5.1. ¿En qué consiste la Gestión de Proyectos?

La gestión de proyectos es la aplicación del conocimiento, habilidades, herramientas y técnicas a las actividades del proyecto de forma tal de cumplir con los requerimientos del proyecto. La gestión de proyectos se lleva a cabo mediante el uso de procesos tales como: iniciación, planificación, ejecución, control y término. El equipo del proyecto gestiona el trabajo de los proyectos, trabajo que comúnmente implica:
Distintas demandas de: alcance, tiempo, costo, riesgo y calidad. Clientes con diferentes necesidades y expectativas. Requerimientos identificados.

Es importante hacer notar que muchos de los procesos contenidos dentro de la gestión de proyectos son iterativos por naturaleza. Esto se debe, en parte, a la existencia y a la necesidad de una elaboración progresiva de un proyecto durante todo su ciclo de vida; es decir, mientras más sabe acerca del proyecto, mejor será la capacidad para manejarlo.

El término gestión de proyectos se utiliza a veces para describir un enfoque organizacional para el manejo o administración de operaciones continuas. Este enfoque, más correctamente llamado gestión por proyectos, trata los diversos aspectos de las operaciones continuas como proyectos de forma tal de aplicar a estos las técnicas de gestión de proyectos. Aunque contar con una comprensión de la gestión de proyectos es un aspecto crítico para aquella organización que realiza la gestión por proyectos, no está dentro del alcance de esta tesis referirse detalladamente al enfoque en sí.

Como cualquier empresa humana, los proyectos necesitan ser ejecutados y entregados bajo ciertas restricciones. Tradicionalmente, estas restricciones han sido alcance, tiempo y costo. Esto también se conoce como el Triángulo de la Gestión de Proyectos, donde cada lado representa una restricción. Un lado del triángulo no puede ser modificado sin impactar a los otros. Un refinamiento posterior de las restricciones separa la calidad del producto del alcance, y hace de la calidad una cuarta restricción.

La restricción de tiempo se refiere a la cantidad de tiempo disponible para completar un proyecto.

La restricción de costo se refiere a la cantidad presupuestada para el proyecto.

La restricción de alcance se refiere a lo que se debe hacer para producir el resultado final del proyecto.

Estas tres restricciones son frecuentemente competidoras entre ellas: incrementar el alcance típicamente aumenta el tiempo y el costo, una restricción fuerte de tiempo puede significar

un incremento en costos y una reducción en los alcances, y un presupuesto limitado puede traducirse en un incremento en tiempo y una reducción de los alcances.

Entonces, podemos decir que la disciplina de la gestión de proyectos consiste en proporcionar las herramientas y técnicas que permiten al equipo de proyecto (no solamente al gerente del proyecto) organizar su trabajo para cumplir con todas esas restricciones.

Tiempo: El tiempo se descompone para propósitos analíticos en el tiempo requerido para completar los componentes del proyecto que es, a su vez, descompuesto en el tiempo requerido para completar cada tarea que contribuye a la finalización de cada componente. Cuando se realizan tareas utilizando gestión de proyectos, es importante partir el trabajo en pedazos menores para que sean fáciles de seguir.

Costo: El costo de desarrollar un proyecto depende de múltiples variables incluyendo costos de mano de obra, costos de materiales, administración de riesgo, infraestructura (edificios, máquinas, etc.), equipo y utilidades. Cuando se contrata a un consultor independiente para un proyecto, el costo típicamente será determinado por la tarifa de la empresa consultora multiplicada por un estimado del avance del proyecto.

Alcance: Requerimientos especificados para el resultado final. La definición global de lo que se supone que el proyecto debe alcanzar y una descripción específica de lo que el resultado final debe ser o debe realizar. Un componente principal del alcance es la calidad del producto final. La cantidad de tiempo dedicado a las tareas individuales determina la calidad global del proyecto. Algunas tareas pueden requerir una cantidad dada de tiempo para ser completadas adecuadamente, pero con más tiempo podrían ser completadas excepcionalmente. A lo largo de un proyecto grande, la calidad puede tener un impacto significativo en el tiempo y en el costo (o viceversa).

2.5.2. Gestión de proyectos de desarrollo de software

En la actualidad persisten problemas en el desarrollo de software, entre ellos, un inadecuado entendimiento de las necesidades de los usuarios, incapacidad de absorber cambios en los requisitos e insatisfacciones de los clientes por inaceptable o bajo desempeño del software. Las principales causas son la administración insuficiente de requisitos; los problemas que afectan la comunicación; las inconsistencias no detectadas entre requisitos, diseño y programación; las validaciones tardías de requisitos; el enfrentamiento reactivo de riesgos y la propagación de cambios sin control.

Existen diferentes metodologías de gestión de proyectos propuestas por diferentes organizaciones; metodologías formales como PMI (Project Management Institute), SEI (Software Engineering Institute), y metodologías ágiles, como son RUP, SCRUM, MSF, entre otras; que ofrecen herramientas de planificación, análisis y control de los proyectos,

necesarias para una buena administración. Estos métodos son indispensables en las organizaciones; ya que para conseguir un proyecto de software fructífero se debe comprender el ámbito del trabajo a realizar, los riesgos en los que se puede incurrir, los recursos requeridos, las tareas a llevar a cabo, el esfuerzo (costo) a consumir y el plan a seguir.

La gestión formal de proyectos se asienta sobre la dirección del proyecto sobre un plan general con visibilidad y ámbito de certidumbre hasta el final del proyecto. La gestión formal hace hincapié en la necesidad de conocer con el mayor detalle los requerimientos desde el principio para dar rigor al plan del proyecto. Las principales referencias de la gestión formal de proyectos son las asociaciones: PMI (Project Management Institute), IPMA (International Project Management Asociation) y la metodología PRINCE2 (Projects in Controlled Environments). PMI e IPMA son organizaciones que han ido desarrollando estándares, métodos y modelos de certificación profesional.

Por otro lado, la planificación de la gestión ágil es informal (algunos modelos llegan a prohibir el uso de diagramas de Gantt) y solo cubren el ciclo de software que se está elaborando (generalmente un mes). Las principales referencias de la gestión ágil de proyectos son: Scrum, Rational Unified Process (RUP) y Microsoft Solutions Framework (MSF). Scrum es un modelo ágil no centrado en prácticas de programación como XP, sino en prácticas de gestión. Rational Unified Process es un proceso iterativo para desarrollo de software creado por Rational Software (IBM). MSF es un marco de desarrollo que define procesos, principios, modelos, disciplinas, conceptos y prácticas contrastadas por Microsoft. Cabe aclarar que estos no son modelos de procesos sino marcos de trabajo adaptables a las circunstancias de las organizaciones de los proyectos.

Muchas organizaciones exitosas hoy día, utilizan una combinación de metodologías formales e informales de desarrollo, y una mezcla de herramientas, para conducir proyectos de manera predecible, que aceleren el desarrollo y reduzcan los riesgos innatos en estos.

Según Métrica-3: “La gestión de proyectos tiene como finalidad principal la planificación, el seguimiento y control de las actividades y los recursos humanos y materiales que intervienen en el desarrollo de un Sistema de Información. Como consecuencia de este control es posible conocer en todo momento qué problemas se producen y resolverlos o paliarlos de manera inmediata”.

Si bien es importante detallar en su totalidad la Gestión y Control de proyectos de software, el objetivo principal de esta tesis es centrarse únicamente en los aspectos de planificación, seguimiento y control de la planificación de los proyectos de desarrollo de software (dejando de lado la estimación) pues el Control representa el segmento o capa superior de la metodología propuesta.

2.5.3. Gestión de proyectos formales

2.5.3.1. PMI

La Guía del PMBOK® es un estándar en la gestión de proyectos desarrollado por el Project Management Institute (PMI). Se encuentra disponible en 11 idiomas: inglés, español, chino simplificado, ruso, coreano, japonés, italiano, alemán, francés, portugués de Brasil y árabe.

En 1987, el PMI publicó la primera edición del PMBOK® en un intento por documentar y estandarizar información y prácticas generalmente aceptadas en la gestión de proyectos. La edición actual, la tercera, provee de referencias básicas a cualquiera que esté interesado en la gestión de proyectos. Posee un léxico común y una estructura consistente para el campo de la gestión de proyectos.

La Guía del PMBOK es ampliamente aceptada por ser el estándar en la gestión de proyectos, sin embargo existen algunas críticas: La mayor viene de los seguidores de la Cadena Crítica (en oposición al Método de la ruta crítica).

El PMBOK es una colección de procesos y áreas de conocimiento generalmente aceptadas como las mejores prácticas dentro de la gestión de proyectos. El PMBOK en un estándar reconocido internacionalmente [IEEE Std 1490-2003] que provee los fundamentos de la gestión de proyectos que son aplicables a un amplio rango de proyectos, incluyendo construcción, software, ingeniería, etc.

El PMBOK reconoce 5 procesos básicos y 9 áreas de conocimiento comunes a casi todos los proyectos. Los conceptos básicos son aplicables a proyectos, programas y operaciones. Los cinco grupos de procesos básicos son:

Inicio,
Planificación,
Ejecución,
Control y Monitoreo, y
Cierre.

Los procesos se traslapan e interactúan a través de un proyecto o fase. Los procesos son descritos en términos de: Entradas (documentos, planes, diseños, etc.), Herramientas y Técnicas (mecanismos aplicados a las entradas) y Salidas (documentos, productos, etc.).

Las nueve áreas del conocimiento mencionadas en el PMBOK son:

Gestión de la Integración de Proyectos,
Gestión del Alcance en Proyectos,

Gestión del Tiempo en Proyectos,
Gestión de la Calidad en Proyectos,
Gestión de Costos en Proyectos,
Gestión del Riesgo en Proyectos,
Gestión de Recursos Humanos en Proyectos,
Gestión de la Comunicación en Proyectos, y
Gestión de la Procura (Logística) en Proyectos.

2.5.3.2. SEI

Software Engineering Institute (SEI) es un instituto federal estadounidense de investigación y desarrollo, fundado por el Congreso de los Estados Unidos en 1984 para desarrollar modelos de evaluación y mejora en el desarrollo de software, que dieran respuesta a los problemas que generaba al ejército estadounidense la programación e integración de los sub-sistemas de software en la construcción de complejos sistemas militares. Financiado por el Departamento de Defensa de los Estados Unidos y administrado por la Universidad Carnegie Mellon.

Es un referente en Ingeniería de Software por realizar el desarrollo del modelo SW-CMM [1991] que ha sido el punto de arranque de todos los que han ido formando parte del modelo que ha desarrollado sobre el concepto de capacidad y madurez, hasta el actual CMMI.

SEI alberga también a otro instituto federal de investigación y desarrollo 3: CERT, fundado por SEI en noviembre de 1988 por encargo de DARPA (Defense Advanced Research Projects Agency) para investigar y mejorar la seguridad de los sistemas de información del ejército y ejercer la coordinación en caso de emergencias.

2.5.4. Gestión de proyectos ágiles

2.5.4.1. Scrum como framework ágil de administración de Proyectos

SCRUM es una metodología ágil de gestión de proyectos cuyo objetivo primordial es elevar al máximo la productividad de un equipo. Reduce al máximo la burocracia y actividades no orientadas a producir software que funcione y produce resultados en periodos muy breves de tiempo (cada 30 días). Sólo abarca prácticas de gestión sin entrar en las prácticas de desarrollo como puede hacer XP. Más bien delega completamente en el equipo la responsabilidad de decidir la mejor manera de trabajar para ser lo más productivos posibles. Sus raíces teóricas están en las teorías de la auto-organización.

Scrum es un método iterativo e incremental que enfatiza prácticas y valores de “project management” por sobre las demás disciplinas del desarrollo. Al principio del proyecto se define el “Product Backlog”, que contiene todos los requerimientos funcionales y no funcionales que deberá satisfacer el sistema a construir. Los mismos estarán especificados

de acuerdo a las convenciones de la organización ya sea mediante: features, casos de uso, diagramas de flujo de datos, incidentes, tareas, etc. El Product Backlog será definido durante reuniones de planeamiento con los stakeholders. A partir de ahí se definirán las iteraciones, conocidas como Sprint en la juerga de Scrum, en las que se irá evolucionando la aplicación evolutivamente. Cada Sprint tendrá su propio "Sprint Backlog" que será un subconjunto del Product Backlog con los requerimientos a ser construidos en el Sprint correspondiente. La duración recomendada del Sprint es de 1 mes.

Dentro de cada Sprint el Scrum Master (equivalente al Líder de Proyecto) llevará a cabo la gestión de la iteración, convocando diariamente al Scrum Daily Meeting que representa una reunión de avance diaria de no más de 15 minutos con el propósito de tener realimentación sobre las tareas de los recursos y los obstáculos que se presentan. Al final de cada Sprint, se realizará un Sprint Review para evaluar los artefactos construidos y comentar el planeamiento del próximo Sprint.

La intención de Scrum es la de maximizar la realimentación sobre el desarrollo pudiendo corregir problemas y mitigar riesgos de forma temprana. 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. Cabe mencionar que Scrum no propone el uso de ninguna práctica de desarrollo en particular; sin embargo, es habitual emplearlo como un framework ágil de administración de proyectos que puede ser combinado con cualquiera de las metodologías mencionadas.

3. METODOLOGÍA

3.1 Introducción

En proyectos de desarrollo de software en ambientes de trabajo virtuales como por ejemplo en desarrollos de software distribuido, muchas veces las factorías no encuentran la metodología más adecuada ya sea para su estilo organizacional o por la falta de control y de herramientas claves que ayuden a los grupo de analistas, diseñadores y programadores diseminados en diferentes partes geográficas pues terminan por hacer o diseñar una metodología propia, algo que por supuesto no está mal, siempre y cuando cumpla con el objetivo. Una metodología consiste en concretar el tipo de ciclo de vida que se va a seguir, la forma en la que se realizan las actividades dentro de cada etapa y el control de los procesos que se ejerce en cada etapa. Si analizamos detenidamente el ciclo de vida del desarrollo del producto, cada etapa tiene datos de entrada y resultados específicos, las cuáles presentan riesgos que son importantes atacar a tiempo. Es por eso la importancia del estudio de los diferentes ciclos de vida y de los controles adecuados que se deben ejercer en cada uno de los procesos, todo sustentado por una correcta Gestión del Proyecto donde se organiza y coordina los pasos a seguir en el proceso de producción del software.

No existe una metodología universal para hacer frente con éxito a cualquier proyecto de desarrollo de software. Toda metodología debe ser adaptada al contexto del proyecto: recursos técnicos y humanos, tiempo de desarrollo, tipo de sistema, etc.

Históricamente, las metodologías tradicionales en el campo de desarrollo de software, han intentado abordar la mayor cantidad de situaciones de contexto del proyecto, exigiendo un esfuerzo considerable para ser adaptadas, sobre todo en proyectos pequeños y con requisitos muy cambiantes. En su lugar las metodologías ágiles emergen como una posible respuesta para llenar ese vacío metodológico. Por estar especialmente orientadas a proyectos pequeños y a la gestión de riesgos, las metodologías ágiles constituyen una solución a la medida para ese entorno.

Dirigir un proyecto en ambientes de trabajo virtuales requiere un método de trabajo muy particular. El acercamiento no difiere demasiado de las metodologías clásicas, aunque tiene en cuenta las especificidades y las características particulares inherentes al contexto. A este respecto, uno de los elementos clave consiste en escoger, entre toda la gama de metodologías disponibles para los encargados de los sistemas de información, la que mejor se adapte a este tipo de proyecto.

Durante la era posterior a la burbuja tecnológica, los presupuestos de TI se recortaron más que las demandas de sus servicios, lo que motivó que los responsables buscarán soluciones más rentables e impulsaran la tendencia de subcontratar el desarrollo de software en países de mercados emergentes (desarrollo internacional). La motivación económica no es el

único factor que influye en esta tendencia. El reciente crecimiento rápido motivado por una infraestructura de comunicaciones mejorada también desempeña un papel importante.

Con respecto al trabajo en equipos distribuidos en general y la subcontratación internacional en particular, el software de voz sobre IP (VoIP), la mensajería instantánea, los clientes de correo electrónico y los wikis[19] que se pueden utilizar han facilitado las comunicaciones en línea. Además, ahora se suele dar preferencia a herramientas en línea, como los wikis, frente a las comunicaciones personales, porque no solo ayudan a transmitir la información, sino que también contribuyen a estructurarla y almacenarla. Estas herramientas también resultan efectivas a la hora de distribuir información a muchos destinatarios.

Estas rápidas conexiones a Internet, disponibles globalmente, impulsan otras herramientas, que se suman a la tendencia. Las herramientas de modelado contribuyen a hacer la documentación más comprensible en equipos distribuidos. Los rastreadores de errores, servidores de control de código fuente, los portales web y las herramientas de colaboración en línea ayudan a coordinar los proyectos distribuidos. Los servicios de Terminal Server y las máquinas virtuales facilitan las pruebas y la administración remotas.

Internet también permitió que las altas tecnologías estuvieran al alcance de los países de mercados emergentes. Dado que Internet traspasa las fronteras políticas, miles de jóvenes de países en vías de desarrollo, utilizan la red de redes para aprender las tecnologías de vanguardia y mejoran sus conocimientos de inglés. Esta nueva ola de ingenieros de software instruidos en Internet vino a reforzar justo a tiempo la tendencia de internacionalización.

3.2 ¿Cuándo se considera que un proyecto de software es exitoso?

Pressman [2002] señala que estos se consideran exitosos cuando elaboran un producto que satisface las necesidades de los usuarios y se mantienen dentro de los plazos y costos presupuestados. Este mismo autor señala que, para tener éxito al diseñar y construir software, es necesario aplicar una disciplina.

¿Dónde se puede obtener una disciplina para diseñar y elaborar software?

La ingeniería de software es un área del conocimiento, que ofrece métodos y técnicas para desarrollar y mantener software de calidad, abordando todas las fases del ciclo de desarrollo de cualquier tipo de sistema de información. Esta disciplina también procura utilizar racionalmente los recursos tratando de establecer principios y métodos a fin de obtener un producto rentable [Bauer, 1972].

19 Wikis: un wiki, o una wiki, es un sitio web cuyas páginas web pueden ser editadas por múltiples voluntarios a través del navegador web

Adicionalmente Wikipedia [2009] señala que esta disciplina incorpora conocimientos de diversas áreas del saber, como lo son: la gestión de proyectos, la gestión de calidad, la ingeniería de sistemas, las matemáticas, entre otras. Pressman [2002] afirma que esta disciplina también comprende las tecnologías aplicadas al proceso de elaboración y mantención, como lo son los métodos técnicos y las herramientas automatizadas. Finalmente, también señala que ésta es una disciplina joven, en permanente evolución y que presenta diversas tendencias. Por ejemplo, algunas corrientes presentes en el área de los modelos de procesos son: Waterfall, Agile, RUP, Spiral, RAD, XP, Cleanroom, Scrum, MSF, entre otras.

Sin embargo, al ser una disciplina joven que presenta un amplio conjunto de tendencias, puede surgir la siguiente interrogante:

¿Qué tendencia debe aplicarse para un exitoso desarrollo distribuido de software?
La respuesta no es trivial, por este motivo la presente investigación provee información para poder tomar una decisión adecuada al momento de establecer un desarrollo remoto.

Toda actividad se facilita cuando se sigue un curso de acciones predecibles. Por este motivo la ingeniería de software proporciona una serie de procesos, métodos y herramientas para que el equipo desarrollador pueda adaptarlas a sus necesidades, dependiendo principalmente del tipo de software que se requiera construir.

- Las herramientas proporcionan un enfoque automático o semi-automático para los procesos y los métodos de desarrollo.
- Los métodos, indican cómo se debe construir técnicamente el software, comprendiendo una gran gama de actividades.
- Los procesos de desarrollo, vinculan a las metodologías con las herramientas y permiten una elaboración racional y oportuna.
- Enfocarse en la calidad, se relaciona con entregar un producto que satisface plenamente las necesidades del cliente basándose en un diseño de calidad y una elaboración que se ajusta a dicho diseño.

Un marco de trabajo común para el proceso de desarrollo, definido por actividades aplicables a todos los proyectos (independientemente al área de aplicación, tamaño y complejidad), se representa de la siguiente manera:

- El conjunto de tareas (Task Sets): Comprende todas las actividades que pueden ser adaptadas a las características del proyecto y a los requisitos del equipo desarrollador. Como lo son las tareas y productos de trabajo, los hitos del proyecto y los puntos de aseguramiento de la calidad.

- Las actividades de protección (Umbrella activities): Son independientes al proyecto y es recomendable aplicarlas durante todo el proceso de desarrollo. Entre las actividades típicas de esta categoría se incluyen:
    - Seguimiento y control del proyecto
    - Revisiones técnicas formales
    - Garantía de la calidad del software
    - Gestión de la configuración
    - Preparación y producción de documentos
    - Gestión de reutilización
    - Mediciones
    - Gestión de riesgo

Como se ha señalado anteriormente, desde los comienzos de la ingeniería de software se han propuesto distintos modelos de procesos. Sin embargo, la elección dependerá de diversos factores, entre ellos:

- La naturaleza del proyecto y su aplicación
- Los métodos y las herramientas que se utilizarán para su desarrollo
- Los controles y entregas que se requieran

Algunos modelos de procesos (o metodologías de desarrollo de software) son:

- El modelo lineal secuencial
- El modelo de construcción de prototipos
- El modelo de desarrollo rápido de aplicaciones
- El modelo espiral
- El modelo de desarrollo concurrente
- El modelo de desarrollo basado en componentes

Una de los objetivos de esta investigación es proponer un modelo de proceso o metodología de desarrollo de software, que incluya un conjunto de tareas y actividades adecuadas para aplicarlas a un desarrollo distribuido de software.

3.3 Requisitos tecnológicos esenciales para logar un exitoso desarrollo distribuido de software

Bellagio y Milligan [2005] platean los siguientes requisitos tecnológicos esenciales para logar un exitoso desarrollo distribuido de software:

- Controlar, medir y dirigir lo que está sucediendo en el proceso de desarrollo.
- Identificar y controlar la documentación, el código y las interfaces.
- Recolectar datos que apoyen la gestión.
- Armar el sistema final utilizando listas que señalen las partes y sus versiones.

- Tener acceso rápido y fácil para encontrar y extraer artefactos, tanto del proyecto actual como de anteriores.
- Evitar la pérdida y la falta de identificación de las versiones de los artefactos mediante la utilización de un método efectivo para organizarlos y localizarlos.
- Los repositorios deben tolerar las fallas, ser escalables, distribuibles y replicables.
- Control de acceso para modificar o visualizar los artefactos.
- Registrar información que permita: la corrección rápida de errores, evitar la incorporación de nuevos errores, y prescindir de cierta comunicación presencial. Por lo cual se debe registrar la siguiente información:
    - ¿Qué fue modificado?
    - ¿Dónde se modificó?
    - ¿Quién produjo los cambios?
    - ¿Cuándo fueron hechos?
    - ¿Por qué fueron hechos?
- Organizar los archivos y directorios para:
    - Reducir la complejidad de su manejo
    - Facilitar la identificación de su calidad
    - Facilitar el intercambio y la reutilización de componentes
    - Ayudar a mantener la arquitectura del software
    - Poder definirlos, controlarlos y gestionarlos jerárquicamente
- Manejar la velocidad de cambio de la elaboración del software.
- Reunir en un mismo lugar y tiempo las versiones adecuadas de los componentes para facilitar su conexión al construir la versión final del software.
- Reproducir los releases anteriores.
- Presentar informes de los avances de desarrollo.
- Registrar y rastrear las solicitudes de cambio para verificar su progreso.
- Priorizar y establecer fechas para materializar las solicitudes de cambio.
- Trazabilidad para vincular y hacer seguimiento de los requisitos, casos de prueba, planificación y artefactos.
- Tener un modelo de decisión para aceptar o rechazar las solicitudes de cambio.
- Implementar consistentemente los cambios para evitar que los integrantes incurran en errores.
- Facilitar el trabajo simultáneo y la recolección de información clave que agilice el proceso de integración.
- Garantizar la consistencia de la configuración, al margen del ambiente de trabajo y construcción.
- Vincular la gestión del proyecto con la gestión de configuración y la gestión de solicitudes de cambio.
- Proporcionar ambientes de trabajo estables.
- Sincronizar los cambios que realizan periódicamente los miembros del equipo.

- Posibilitar el trabajo aislado.
- Aumentar el nivel de abstracción de la comunicación desde archivos a actividades, porque es una forma de comunicación más similar a la efectuada por las personas.
- Aislar los cambios disruptivos para no afectar la integridad del proyecto y de los otros miembros.
- Proporcionar información actualizada y acceso a los últimos artefactos elaborados en el proyecto.
- Poder modificar, integrar y fusionar los cambios realizados, en un tiempo apropiado y minimizando la sobrecarga de trabajo.
- Registrar, cómo se construyó el software y las versiones de los componentes que lo conforman.
- Mantener el control sin que influya la cantidad de cambios realizados en el software y productos de trabajo.
- El jefe de proyecto debe tener acceso rápido y actualizado de las actividades asignadas a cada integrante.
- Posibilitar el desarrollo paralelo y los cambios concurrentes de los artefactos.

Para Bellagio y Milligan [2005], estos requisitos son satisfechos por un conjunto de herramientas y procesos adecuados, los que permiten minimizar el riesgo inherente al desarrollo distribuido de software.

3.4 Una Metodología flexible

Podemos comenzar diciendo que los proyectos clásicos tienen algunas dificultades que han llevado a criticar cada vez más las metodologías clásicas -denominadas "de cascada"- debido al encadenamiento sucesivo de etapas bien delimitadas (análisis de las necesidades, diseño, especificaciones, etc.). Normalmente se les atribuyen dos grandes inconvenientes: por un lado, la dificultad para adaptarlas a especificaciones variables que, a menudo, son inevitables en los proyectos de larga duración y, por otro lado, la dificultad que implica la fase de integración, en la que es necesario sincronizar todos los módulos desarrollados por equipos independientes de desarrollo.

El análisis de estas metodologías, con el tiempo, ha dado lugar a un nuevo tipo, las denominadas metodologías ágiles. Su característica principal es que utilizan ciclos de iteración cortos a lo largo de todo el proyecto y encadenan rápidamente todas sus etapas, desde el planteamiento inicial hasta las pruebas funcionales del módulo desarrollado. La duración de los ciclos puede variar, pero suele ser de unos quince días. Esto conlleva claras ventajas: por un lado, permite adaptarse rápidamente a los cambios y variaciones en las especificaciones, sobre todo los que se derivan del análisis de los resultados realizado por el cliente al final del ciclo; y por otro lado, se simplifica notablemente la integración, ya que se va realizando a medida que avanzan los ciclos. Así pues, el análisis que se hace a la hora

de trabajar en proyectos de desarrollo de software en ambientes virtuales cuyos integrantes se encuentran dispersos geográficamente, es utilizar este tipo de metodología para simplificar toda una serie de problemas que suelen surgir en este contexto.

Las técnicas Ágiles en realidad ayudan a hacer frente y mitigar los problemas habituales de Proyectos de desarrollo de software distribuido: la falta de visibilidad sobre el estado del proyecto, retraso en la retroalimentación del ciclo, la pérdida del negocio y el contexto técnico, disminución en la comunicación, mayor documentación, y desconfianza.

Las iteraciones cortas, con una demostración del avance del producto, aumenta la visibilidad del estado del proyecto y proporciona una retroalimentación instantánea, dando como oportunidad a un proceso de ajuste del producto de manera incremental a lo largo del tiempo de ejecución del Proyecto; de esta manera facilita la comunicación entre el cliente conocedor del negocio y la gente que participa en el desarrollo del Proyecto.

3.5 Comunicación

Al trabajar de forma remota, los pequeños malentendidos se convierten rápidamente en problemas mayores. En los equipos de desarrollo distribuidos, los responsables deben prestar atención a las prácticas de comunicación que a veces omiten sin que tenga consecuencias negativas en el desarrollo local. Esta atención incluye la creación de informes periódicos (diarios/semanales) y reuniones de actualización de estado, que permitan a los miembros del equipo sincronizarse, tratar los logros conseguidos y poner de manifiesto los problemas. Los responsables también deben tratar de fomentar las relaciones personales en los equipos mediante reuniones de presentación, visitas a oficinas, actividades de creación de equipos y otras prácticas.

En tratos de subcontratación internacional, los responsables de desarrollo deben conocer el idioma, la cultura y las diferencias horarias, y deben encontrar formas de superar estos obstáculos. La globalización borra constante pero lentamente las distinciones culturales en el entorno profesional, aunque todavía hay casos en los que las diferencias culturales provocan confusión. Los problemas relacionados con el idioma son mucho más fáciles de detectar, aunque no necesariamente tienen más fácil solución. En los casos en que las compañías deben enfrentarse a la barrera idiomática, es habitual y sumamente deseable impartir cursos de formación entre los empleados. En la mayoría de los países en vías de desarrollo, los profesionales están motivados para aprender inglés, de modo que normalmente estas personas son las que obtienen formación en un idioma.

Las diferencias entre zonas horarias dificultan especialmente el proceso. Pero resulta que en países con sectores de subcontratación desarrollados, los ingenieros de software generalmente están dispuestos a adaptar su horario laboral al de sus homólogos extranjeros. Hay dos estrategias para tratar las diferencias entre zonas horarias. La primera consiste en

separar los equipos por actividad; por ejemplo, tener a los responsables de control de calidad y producto en la oficina y a los desarrolladores en el extranjero. Esta distribución permite implementar un ciclo en el que los desarrolladores se ocupan de implementar correcciones y requisitos nuevos mientras sus homólogos están durmiendo y viceversa. Por supuesto, debe haber un momento del día en que los horarios de trabajo coincidan (por ejemplo, al comienzo o al final de la jornada laboral). El segundo enfoque consiste en dividir los proyectos en bloques e intentar asignar cada uno a una ubicación, delegando tantas funciones como sea posible en esta ubicación. El segundo enfoque favorece una mejor comunicación y contribuye así a facilitar el desarrollo ágil, pero ambos funcionan y, en ocasiones, no hay elección.

Elegir el modelo correcto también es muy importante. Se recomienda que al menos una de las partes implicadas tenga experiencia en el desarrollo ágil, preferiblemente en un entorno distribuido. La ausencia de comunicación cara a cara, junto con las diferencias horarias, culturales e idiomáticas, requieren atención y esfuerzos adicionales con el fin de obtener los resultados deseados. Las ventajas de tener un buen socio comercial internacional que permita ahorrar costos, incrementar el personal según la demanda y subcontratar tareas relacionadas con la infraestructura (lo que se podría resumir como "obtener más por menos") superan de lejos la inversión que supone la creación de relaciones productivas. Este balance positivo no sería posible sin herramientas modernas respaldadas por la excepcional infraestructura de comunicaciones disponible en todo el mundo.

3.6 Utilizar la arquitectura de software para minimizar los requisitos de comunicación y colaboración

Esta práctica plantea que los proyectos deben organizarse en torno a la arquitectura de software para facilitar la comunicación y producir una colaboración efectiva. El problema se produce cuando el tamaño del proyecto es mayor, porque la comunicación entre los miembros se vuelve más compleja, lo que aumenta su costo y dificulta que los miembros conozcan la globalidad del proyecto. En consecuencia, se deben minimizar las necesidades de comunicación y para esto se plantea organizar al equipo en torno a la arquitectura porque permite reducir significativamente los canales de comunicación [Kroll y Maclsaac, 2006].

Uno de los beneficios de una arquitectura de software robusta, consiste en poder dividir claramente las responsabilidades del sistema en subsistemas bien definidos, con interfaces bien definidas. De esta manera, al organizarse en función de la arquitectura se reduce el riesgo de trabajo duplicado.

En cuanto a los problemas relativos a la interacción de los subsistemas, éstos deben resolverse por el equipo de arquitectos, quienes son los que proveen las interfaces entre estos.

Para su adecuada aplicación, los arquitectos deben trabajar con todo los integrantes del proyecto para garantizar que se detallen los requisitos fundamentales que afecten la arquitectura, diseño, implementación y pruebas [Ambler y Jeffries, 2002].

Para determinar la estructura del equipo desarrollador, por su parte, existen diversas formas para dividir una aplicación en subsistemas, por ejemplo; en torno a las funciones o a las características horizontales y verticales del negocio.

Es así como cada subsistema debe tener algún equipo responsable que comprenda las restricciones de la arquitectura, las interfaces entre subsistemas, las elecciones tecnológicas, los patrones arquitectónicos que deben incorporarse y los requisitos arquitectónicos. Por lo general, estos mismos equipos deben realizar cambios o ajustes arquitectónicos que afecten el comportamiento y/o la interfaz de los subsistemas, es por esto que los equipos encargados deben trabajar en conjunto con el equipo de arquitectura del proyecto, para garantizar que los cambios requeridos sean aceptables no sólo a nivel del subsistema sino también en su totalidad. Cuando los cambios son aceptados, deben comunicarse a todos los miembros del proyecto. Kroll y Maclsaac [2006] recalcan que el equipo de arquitectura y el equipo responsable por el subsistema no deben trabajar de forma aislada, sino mediante estrecha colaboración para garantizar una correcta elaboración.

En relación a promover la propiedad colectiva, donde cualquier miembro esté facultado para realizar cambios en el código o en la modelación, como se señala en XP [Beck y Andrew, 2004] y Agile Modeling [Ambler y Jeffries, 2002], Kroll y Maclsaac [2006] expresan que esta práctica funciona sólo para proyectos pequeños de limitada complejidad, donde los integrantes pueden comunicarse fácilmente para entender que cambios han sido hechos y el porqué; o por equipos ligeramente mayores, pero que poseen miembros muy talentosos.

Para la mayoría de los proyectos y especialmente los mayores, es necesario asegurar que los responsables de los subsistemas serán notificados ante cualquier cambio, de otra manera se corre el riesgo de introducir problemas inesperados en el código. Al respecto se recomiendan que en los proyectos largos y/o complejos los miembros encargados de los subsistemas sean los que efectúen los cambios.

3.7 En un mundo virtual, las personas y sus relaciones ante todo

Una de las características más importantes a la hora de conformar un equipo de trabajo para el desarrollo de software en ambientes dispersos geográficamente, es el perfil de las personas que integran dichos equipos. Para que los equipos de desarrollo trabajen de manera mancomunada, más allá de las distancias que los separa geográficamente, es menester contar con personas con ciertas características en su personalidad. Al respecto,

Skyrme [1999] considera que la puesta en práctica del trabajo virtual se fundamenta en 25 principios de práctica probada, que se reflejan a continuación:

Pre-requisitos (actitudes y comportamientos individuales)

- Todo individuo debe autovalorarse y debe valorar a cualquier otro miembro del equipo por sus contribuciones, que deberían ser explícitas y expresadas como las capacidades distintivas del equipo. Los individuos deberían aprender unos de otros, del resultado de sus propias acciones y de la experiencia colectiva.

- Debe haber un mayor nivel de confianza. Esto puede suponer mucho tiempo, pero el punto de partida es confiar en todas las personas hasta que abusen de esta confianza.

- Los individuos deben apoyarse mutuamente y deberían alcanzarse los compromisos. En caso de que las circunstancias impidiesen lograrlos, los otros miembros del equipo deben ser informados tan pronto como sea posible.

- Debe prevalecer la reciprocidad. Dar tanto como se recibe, en términos de apoyo, transferencia de información y conocimiento. La falta de reciprocidad conduce a relaciones desequilibradas y finalmente a la jerarquía, la retirada o el fracaso del equipo.

- Se deben reconocer y expresar los sentimientos particulares. Compartir esto es una buena forma de iniciar y concluir los encuentros de los equipos.

Equipos (composición)

- Los equipos son las unidades organizativas que crean ‘focos’ y permiten que el trabajo continúe. Es preciso trabajar en equipo e individualmente si uno desea continuar desarrollando su conocimiento y éxito.

- Los equipos más productivos para el conocimiento funcionan como pequeños grupos multidisciplinares (ejemplo, de cinco a ocho personas con variada experiencia y rasgos de personalidad).

- Los equipos numerosos no son productivos para el trabajo de conocimiento. Son reuniones o comisiones que pueden utilizarse para pasar información (a menudo, ineficazmente), motivar (o desmotivar) y proporcionar una sensación de importancia. Su uso más valioso es crear y mantener una sensación de pertenencia, cohesión y reforzamiento, y, por supuesto, de oportunidades de trabajo en red.

- Cada empleado de conocimiento debería pertenecer al menos a dos equipos distintos. Esto ayuda a que la organización logre la cooperación inter-funcional y que los individuos obtengan una perspectiva más amplia.

- Un individuo puede desempeñar diversos roles en el equipo. Estos roles pueden cambiar y ser intercambiados (por ejemplo durante los periodos de vacaciones, para equilibrar trabajo o para ampliar la experiencia individual). Se debe distinguir el rol de cada persona.

Normas y relaciones del equipo (misión, propósito y cultura)

- Todo equipo debe tener un propósito si quiere actuar como tal y no como una reunión de individuos. Debe tener su propia visión, misión y objetivos que refuercen los del nivel superior.

- Todos los equipos deberían desarrollar unas normas y valores culturales fuertes. Por lo tanto, deberían llevarse a cabo regularmente reuniones del equipo. Se debería desarrollar una serie de principios de trabajo.

- Cada equipo debería identificar a otros equipos que realizan actividades relacionadas o dependientes. Debería dibujar un diagrama de la red: el equipo (y su misión) en el centro, un anillo cercano de los equipos (nodos) con altas interdependencias (relaciones formales) y un anillo lejano de equipos colaboradores (principalmente que comparten información). Se debe mostrar, donde fuera posible, la secuencia de actividades principales e interdependencias (quién proporciona qué a quién).

- Se debería animar a los miembros de los equipos a mantener sus redes particulares/personales, incluso más allá de las necesidades identificables del proyecto o equipo actual. Las redes profesionales y externas son particularmente importantes.

- La red debe construirse con cierta flexibilidad. Una cierta duplicidad o solapamiento no debería verse como algo negativo. Esta flexibilidad permite una mayor calidad de resultado y una capacidad de resistencia para afrontar lo inesperado.

Comunicaciones

- Como en las redes electrónicas, se necesita definir y acordar una serie de protocolos. Pueden ser implícitos (estándares comunes determinados por los valores culturales), aunque a menudo es necesario explicitar lo que significan las diversas

señales (por ejemplo idea, acción, demanda, decisión). Probablemente la mala comunicación es el peor obstáculo para la eficacia de cualquier organización.

- Se debe potenciar una comunicación frecuente a través de toda la red (incluyendo el anillo externo). Esto es especialmente valioso para ideas mal concebidas y posiciones provisionales. Un grupo pequeño que desarrolla su propio comunicado no fomenta el espíritu de la red.

- También como en la comunicación electrónica, el hecho de que un nodo no responda es una señal importante.

- Las relaciones formales están mejor consolidadas al disponer de procesos escritos acordados (automáticos) y/o miembros comunes en ambos equipos. Los enlaces importantes requieren una mayor confianza y transparencia más que una mayor formalidad. En una serie secuencial de tareas esto puede lograrse mediante equipos con miembros compartidos.

- Reconocer el carácter imprevisible y la falta de claridad del proceso de toma de decisiones. Quien toma las decisiones a menudo será ambiguo. Una acción realizada podría implicar una decisión tomada. En general, las decisiones deberían tomarse cuando y donde sea preciso, por quienquiera que sea adecuado. Debe guiarse por la misión, los valores y los principios.

Tecnología y trabajo a distancia

- En el correo electrónico, seleccionar adecuadamente la dirección a quien va dirigido el mensaje, utilizar títulos explícitos y ser explícito respecto a la respuesta que se pretende del lector (por ejemplo información, acción, ayuda).

- Utilizar un correo electrónico para cada tema, especialmente cuando están implicados múltiples destinatarios con roles e intereses distintos. Esto permite que cada uno sea archivado y tramitado por separado. Redactar correos cortos.

- Si la conversación cara a cara es importante, el envío de un segundo correo puede reproducir esta esencia. Actúa como un punto de referencia para las partes implicadas. Puede arrojar distintas interpretaciones de la misma reunión y poner de relieve ambigüedades que es preciso resolver. También actúa como parte de la memoria del equipo.

- Agregar conocimiento que existe o que ha sido expresado. Reconocer las contribuciones de los otros. Designar a un editor de conocimiento que obtenga lo

mejor de la información transitoria y lo reúna en un documento más estructurado o en una página web.

- Por encima de todo, ser humano e informal. Los correos electrónicos y las listas de discusión son conversaciones y, si no se está cara a cara, es necesario introducir cierto grado de informalidad y símbolos que describan una sonrisa o cualquier otra expresión facial donde sea apropiado.

Para lograr la exitosa entrega de proyectos de desarrollo de software, Agile considera que el mejor enfoque es integrar proyectos alrededor de gente motivada y después asegurar que cuenten con las herramientas, los procesos y habilidades apropiadas para llevar a cabo el trabajo.
Se recomienda conveniente que a intervalos regulares, el equipo reflexione acerca de cómo ser más efectivo, entonces realiza los ajustes y modifica su comportamiento en consecuencia.

Uno de los objetivos de las metodologías tradicionales es desarrollar un proceso donde las personas involucradas sean partes reemplazables. Con tal proceso se puede tratar a las personas como recursos que están disponibles en varios tipos. Se tienen un analista, algunos programadores, algunos verificadores, un gerente. Los individuos no son tan importantes, sólo los roles lo son. De esa manera si usted planea un proyecto no importa qué analista y qué verificadores consiga, sólo hay que saber cuántos son para saber cómo afecta su plan el número de recursos.
Pero esto plantea una pregunta clave: ¿son las personas involucradas en el desarrollo de software partes reemplazables? Uno de los rasgos importantes de los métodos ágiles es el rechazo a esta afirmación.

Alistair Cockburn [2000], en su artículo “Caracterización de las Personas como Componentes No Lineales de Primer Orden en el Desarrollo de Software” [Conferencia Sistemas, Cibernética e Informática, Orlando, EEUU, 2000], apunta a que los procesos predecibles requieren componentes que se comporten de manera predecible. Sin embargo las personas no son componentes predecibles. Además sus estudios de proyectos de software lo han llevado a concluir que la gente es el factor más importante en el desarrollo de software.

Uno se pregunta si la naturaleza del desarrollo de software funciona contra uno aquí. Cuando programamos una computadora, controlamos un dispositivo inherentemente predecible. Como estamos en este negocio porque somos buenos en hacerlo, estamos idealmente hechos para perturbarnos cuando nos enfrentamos con seres humanos.
Aunque Cockburn es el más explícito en su visión centrada en la gente del desarrollo de software, la noción de que la gente es primera es un tema común para muchos pensadores

del software. El problema, demasiado a menudo, es que la metodología se ha opuesto a la noción de las personas como el factor de primer orden en el éxito del proyecto.

Esto crea un fuerte efecto de retroalimentación positiva. Si se espera que todos sus desarrolladores se junten en unidades de programación compatibles, no se intentará tratarlos como individuos. Esto baja la moral (y la productividad). Las personas capaces se buscarán un lugar mejor donde estar, y acabará con lo que se deseaba: unidades de programación compatibles. Decidir que las personas son primero es una gran decisión, que requiere mucha determinación. [Cockburn, A., 2000].

Para el Instituto de Ingeniería de Software (SEI), el personal también es un factor clave de la gestión, por tal motivo ha desarrollado un Modelo de Madurez de la Capacidad de Gestión del Personal (PM-CMM), definiendo las siguientes áreas: reclutamiento; selección; desarrollo de la carrera; y diseño de la organización, trabajo, cultura y espíritu de equipo.

3.8 Metodologías tradicionales y ágiles, mejores prácticas

Una metodología de desarrollo es un la conjunción de métodos que especifican quién debe hacer qué cosa, cuándo debe hacerse, estableciendo un conjunto de roles, actividades y un ciclo de vida que establece las diferentes fases.

El desarrollo de software es riesgoso y difícil de controlar, más aún en ambientes de desarrollo virtuales dispersos geográficamente. Prueba de ello es que existen muchas metodologías que inciden en aspectos del proceso de desarrollo. Por una parte tenemos aquellas más “tradicionales”, que llevan asociado un marcado énfasis en el control del proceso, mediante una rigurosa definición de roles, actividades y artefactos, incluyendo modelado y documentación detallada. Este esquema tradicional para abordar el desarrollo de software, ha demostrado ser efectivo y necesario en proyectos de gran envergadura (respecto a tiempo y recursos), donde por lo general se exige un alto grado de ceremonia en el proceso. Sin embargo, este enfoque no resulta ser el más adecuado para muchos de los proyectos actuales, donde las necesidades de los clientes son muy cambiantes, y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad.

Bajo este escenario, ha surgido una corriente en la industria del software que considera que las necesidades de los clientes son muy cambiantes. En pos de esto surgen metodologías que parecen contradecir la visión tradicional, por estar especialmente orientadas para proyectos pequeños, con plazos reducidos, requisitos volátiles, y/o basados en nuevas tecnologías, por lo que las metodologías ágiles constituyen una solución a medida para ese entorno, aportando una elevada simplificación, que a pesar de ello, no renuncia a las prácticas esenciales para asegurar la calidad del producto.

Figura 5. Prácticas ágiles "transversales" [Miaton, L., 2008]

Lo que estos métodos tienen en común es su modelo de desarrollo incremental (pequeñas entregas con ciclos rápidos), cooperativo (desarrolladores y usuarios trabajan juntos en estrecha comunicación), directo (el método es simple y fácil de aprender) y adaptativo (capaz de incorporar los cambios). Las claves de los métodos ágiles son la velocidad y la simplicidad. De acuerdo con ello, los equipos de trabajo se concentran en obtener lo antes posible una pieza útil que implemente sólo lo que sea más urgente; la prioridad es satisfacer al cliente mediante entregas tempranas y continuas de software que le aporte un valor. En este momento se espera el feedback del cliente y se dan la bienvenida a los cambios solicitados, interpretándolos como un avance en la comprensión y luego satisfacción del cliente.

Por lo dicho podemos concluir que en las metodologías tradicionales, en primer lugar se estudian bien los requisitos del producto para después estimar y planificar las actividades y los recursos necesarios para planificar su ejecución. Y en las metodologías ágiles, por su carácter adaptativo, el producto es realizado progresivamente, entregando partes del software totalmente funcionales en cada una de sus iteraciones, considerando en forma natural los ajustes que surgieran en el transcurso del tiempo.

La gestión de los procesos de desarrollo, sin un adecuado soporte metodológico, llevará a costos y elevados riesgos de gestión, sobretodo si se aplican a proyectos de gran envergadura.

Está de más decir que en la práctica, no existe un proceso de desarrollo universal, aplicable a todo proyecto [Brooks, 1987]. Las características del equipo de desarrollo, el dominio de aplicación, el tipo de contrato, la complejidad y envergadura del proyecto, etc., todos estos factores hacen necesario adaptar nuestra metodología de desarrollo y además es posible utilizar las "Best Practices" en un modo integrado y transversal a la metodología en si misma que se emplee.

Al analizar todo lo anterior encontramos que para establecer una metodología completa que se adapte a procesos de desarrollo de software en ambientes virtuales, es necesario definir en un único contexto controles (gestión), procesos (metodología) y herramientas que soporten, organicen y administren el avance de la producción del producto software.

Para ello la metodología en su totalidad está basado sobre un modelo iterativo de desarrollo con una clara definición de roles y responsabilidades, una definida infraestructura para facilitar la coordinación y la colaboración entre todos los miembros del equipo de desarrollo.

3.9 Prácticas seleccionadas aplicadas a la metodología propuesta

- Minimizar el grado de dispersión geográfica
- Establecer un proceso de ingeniería de requisitos bien definido y comprendido
- Realizar reuniones colaborativas frecuentemente, fijando reglas de estilo y frecuencia:
    - o Reuniones Scrum diarias entre el jefe de proyecto y los miembros remotos, utilizando protocolos de comunicación, por ejemplo enviando las respuestas por mail antes de la reunión
    - o Reuniones diarias en cada sub-equipo
    - o Reuniones Scrum semanalmente entre los arquitectos del proyecto
    - o Reuniones diarias (o lo más frecuentemente posible) que involucren al jefe de proyecto, los analistas y al cliente
    - o Cada dos semanas realizar reuniones de planificación donde se deben solicitar los requisitos
- Mejorar la coordinación y comunicación entre los sitios
    - o Establecer responsabilidades comunicacionales en cada sitio
    - o Incrementar la percepción sobre lo que ocurre en el proyecto, la compañía y los sitios remotos
- Utilizar herramientas y tecnologías que apoyen la accesibilidad, colaboración y ejecución de los proyectos
    - o Utilizar una herramienta para realizar construcciones automáticas varias veces al día
    - o Utilizar una herramienta especial para sustentar una efectiva ingeniería concurrente que mida el progreso del proyecto mediante el seguimiento de funcionalidades, tareas y actividades
    - o Estandarización de herramientas y procesos para los diferentes sitios
    - o Centralizar el acceso a las herramientas mediante la web, bases de datos replicadas, ambientes de desarrollo individuales y repositorios centrales
    - o Rápido acceso a la red de trabajo
    - o Compartir recursos como bases de datos, servidores y repositorios del proyecto

  - o Conectividad fácil y rápida entre los sitios remotos
  - o Tecnologías colaborativas que permitan incrementar la efectividad y satisfacción personal
- Promover vínculos sociales entre los integrantes
  - o Construir relaciones sociales para facilitar la buena comprensión entre los interlocutores y establecer confianza
  - o Incrementar la accesibilidad entre los miembros de los sub-equipos, para conocer la disponibilidad de los otros integrantes (día y hora)
  - o Crear y mantener una atmósfera de equipo, evitando el aislamiento de los sitios remotos
  - o Facilitar y permitir la integración entre los miembros
  - o Facilitar el intercambio de personal entre los sitios para reducir las brechas culturales
- Promover el intercambio de conocimiento
  - o Originar, documentar y divulgar conocimiento propio de cada equipo y promover las capacidades de cada miembro
  - o Ampliar el conocimiento que se posee de los sub-equipos
  - o Capacitar a los miembros del equipo y constantemente ir incorporando nuevas tecnologías
- Promover y facilitar la gestión de componentes
  - o Diseñar los componentes para que sean reutilizados
  - o Invertir en proyectos de investigación y desarrollo
  - o Facilitar la reutilización, identificando las oportunidades que permitan emplear nuevamente los componentes y el conocimiento elaborado
- Prácticas básicas de gestión de la configuración
  - o Identificar y almacenar los artefactos en repositorios seguros
  - o Controlar y auditar los cambios de los artefactos
  - o Mantener ambientes de trabajo de forma estable y consistentemente
  - o Soportar el cambio concurrente de artefactos y componentes
  - o Garantizar la reproducción de las construcciones de software
- Integrar temprana y frecuentemente (práctica avanzada de gestión de la configuración)
- Registrar y rastrear las solicitudes de cambio (práctica avanzada de gestión de la configuración)
- Organizar e integrar un conjunto consistente de versiones utilizando actividades (práctica avanzada de gestión de la configuración)
- Realizar iteraciones cortas (de una o dos semanas), que vayan entregando funcionalidades incrementalmente

3.10 Uso de indicadores para medir la eficacia de la metodología

Una forma de determinar si la metodología que se adoptará está cumpliendo con los objetivos esperados, es contar con un conjunto de indicadores que permitan a los equipos de trabajo evaluar las condiciones que se presentan a fin de determinar si es factible incorporar la metodología propuesta en la construcción de un producto software por medio de equipos de trabajo dispersos geográficamente. Para llevar a cabo dicha acción se deberá evaluar la gestión total del proyecto, comparar los resultados obtenidos con criterios previamente establecidos y hacer un juicio de valor, tomando en cuenta la magnitud y dirección de la diferencia encontrada entre lo previsto y lo obtenido.

Los indicadores, son formulaciones generalmente matemáticas con las que se busca reflejar una situación determinada. De acuerdo con Guardiola [1998], un indicador es una relación entre variables cuantitativas o cualitativas que permite observar la situación y las tendencias de cambios generadas en el objeto o fenómeno observado, en relación con objetivos y metas previstas e impactos esperados. Estos indicadores pueden ser valores, unidades, índices, series estadísticas, etc. Son las herramientas fundamentales de la evaluación.

Guardiola [1998] define y tipifica a los indicadores de la siguiente manera:

Los indicadores son útiles para varios fines:

- Evaluar la gestión
- Identificar oportunidades de mejoramiento
- Adecuar a la realidad objetivos, metas y estrategias
- Sensibilizar a las personas que toman decisiones y a quienes son objeto de las mismas, acerca de las bondades de los programas
- Tomar medidas preventivas a tiempo
- Comunicar ideas, pensamientos y valores de una manera resumida: "medimos lo que valoramos y valoramos lo que medimos"

Un indicador aislado, obtenido una sola vez, puede ser de poca utilidad. En cambio, cuando se analizan sus resultados a través de variables de tiempo, persona y lugar; se observan las tendencias que el mismo puede mostrar con el transcurrir del tiempo y se combina con otros indicadores apropiados, se convierten en poderosas herramientas de gerencia, pues permiten mantener un diagnóstico permanentemente actualizado de la situación, tomar decisiones y verificar si éstas fueron o no acertadas.

3.10.1. Características de un buen indicador

Un buen indicador:

- sirve a un propósito;

- se ha diseñado teniendo en cuenta este propósito y las características de los usuarios;
- guarda relación con un asunto de interés actual o futuro (es decir, es útil);
- es costo-eficaz: logra el objetivo de su utilización con la mínima cantidad de recursos, utiliza recursos (datos, entre ellos) existentes o permite utilizar los datos nuevos que requiere para otros usos y usuarios;
- es válido, es decir que mide lo que se pretende medir;
- es objetivo: permite obtener el mismo resultado cuando la obtención del indicador es hecha por observadores distintos, en circunstancias análogas;
- es sensible: es capaz de captar los cambios ocurridos en la situación objeto del indicador;
- es específico: aplicable solo a la situación de que se trata;
- es inequívoco en su significado;
- se puede obtener sin dificultad;
- es consistente en el transcurso del tiempo;
- se obtiene oportunamente;
- es preciso;
- es transparente (fácilmente entendido e interpretado por los usuarios);
- es dado a conocer periódicamente a las partes interesadas.

Guardiola [1998] destaca que estos criterios tienen varias implicaciones que condicionan y limitan los tipos de indicadores que se pueden desarrollar, y la forma como se pueden construir, presentar y utilizar. Muchos de estos criterios son también en cierto grado mutuamente incompatibles: ésa es una razón por la que los indicadores son difíciles de diseñar. La necesidad esencial de costo-eficacia, por ejemplo, significa a menudo que los indicadores se deben desarrollar con base en los datos que ya existen o, si éstos se van a recoger por primera vez, que puedan ser utilizados también para otros propósitos. La necesidad de claridad y de facilidad de entender también implica que los indicadores deben condensar a menudo grandes volúmenes de datos en un breve resumen (como lo es un indicador), y que las complejidades del mundo se reducen a un mensaje simple e inequívoco. El criterio de validez científica, por otra parte, requiere que el proceso de precisión no vaya demasiado lejos. Los indicadores deben simplificar, sin sesgar, la verdad subyacente, o perder las conexiones y las interdependencias vitales que gobiernan el mundo verdadero. Al mismo tiempo, si los indicadores deben ser sensibles al cambio, es necesario que se basen en datos exactos, de alta resolución y consistentes.

Las diversas aplicaciones que pueden darse a los indicadores también crean desafíos. Cada uso puede implicar la necesidad de un indicador ligeramente distinto. Un indicador ideado para monitorear tendencias en el tiempo, por ejemplo, debe basarse en datos que son representativos espacialmente, pero no necesariamente intensivos o completos. El mismo

indicador, usado para examinar patrones geográficos y para identificar lugares de interés, deberá basarse en datos espaciales que son detallados y comprensivos: las variaciones temporales serán menos importantes.
Los indicadores también deben ser dinámicos. Se deben actualizar y corregir en la medida en que el entorno cambia: cambios no solamente en las condiciones específicas que ellos describen, sino también en la disponibilidad de datos, en el conocimiento científico, o en los niveles de interés y necesidades de sus usuarios.

Los indicadores, por lo tanto, no son fijos ni universales. Lo que hace que un indicador sea bueno en un lugar en un momento determinado no será necesariamente relevante en otro. Por consiguiente, aunque es posible idear conjuntos definitivos de indicadores que responden a necesidades específicas, la utilidad más amplia de éstos es inevitablemente limitada. Por otra parte, no es apropiada una especie de anarquía, en la cual cada uno desarrolla sus propios indicadores. Esto daría lugar a una duplicación esfuerzos, a la proliferación de conjuntos de indicadores y a una dificultad cada vez mayor de comparar o de combinar indicadores provenientes de diversas fuentes. Puede también alentar el desarrollo de indicadores mal concebidos y mal diseñados que pueden desinformar más bien que informar.

3.10.2. Tipos de indicadores

Según como se expresa la valoración, los indicadores pueden ser:

- Nominativos o cualitativos, si solo expresan la presencia o ausencia de una cualidad.
- Cuantitativos, si se expresan en forma numérica (porcentajes, promedios, tasas, etc).

Por su importancia relativa, se pueden clasificar como:

- Esenciales o principales
- Secundarios o complementarios

3.10.3. Modelo propuesto de indicadores

El modelo propuesto contempla la identificación de aquellas variables que mejor describen el entorno, es decir, las características técnicas, ambientales y profesionales que rodean al desarrollo de un proyecto de software distribuido geográficamente.

Para lograr con el objetivo, se tomó como base las experiencias de empresas de desarrollo de software offshore y de las experiencias de Jeff Sutherland [2007] y Martin Fowler [2006]. Se identificaron un conjunto de preguntas que afectan a la productividad en este tipo de proyectos. Estas preguntas fueron agrupadas de acuerdo a lo que sugiere la metodología propuesta en el presente trabajo. Se han definido preguntas de Gestión, de

Desarrollo (Metodológico) y de uso de Herramientas. Además se ha incorporado un grupo adicional, el de Recursos Humanos, pues es parte esencial de lo que propone la metodología.

Estos grupos de preguntas están divididas en preguntas esenciales o principales y preguntas secundarias o complementarias. Las preguntas esenciales están relacionadas concretamente con los equipos de trabajo que se irán a formar para afrontar un proyecto de desarrollo de software distribuido geográficamente. Estas preguntas están relacionadas para medir la capacidad individual de cada integrante ya que debe cumplir con ciertas condiciones particulares a la hora de ser parte de un equipo de trabajo de desarrollo de software disperso geográficamente. En cuanto a las preguntas secundarias o complementarias, sirven para determinar definitivamente si la metodología utilizada en el desarrollo de software distribuido cumple con las exigencias que se aplican a estos tipos de proyectos.

Por otro lado se ha determinado el peso relativo de cada pregunta, estableciéndose como valores de peso los siguientes:

- cumple con lo requerido: 2
- cumple parcialmente con lo requerido: 1
- no cumple con lo requerido: 0

Para determinar si la metodología que se utilizará para el desarrollo de software distribuido geográficamente es la correcta, en primer instancia el valor final resultante de sumar los pesos relativos de las preguntas deberá estar por sobre el promedio del total del grupo para poder pasar a evaluar los indicadores secundarios o complementarios.

Los indicadores secundarios están compuestos por preguntas agrupadas en Gestión, Metodología de desarrollo de software y uso de Herramientas. En este caso se ha establecido que para determinar si la metodología de desarrollo de software distribuido geográficamente es la correcta, el valor resultante final de la suma de los pesos relativos de las preguntas por cada grupo debe ser mayor al promedio, así pues, si la sumatoria de los pesos relativos de un solo grupo es igual o menor al promedio, directamente se deberá descartar la metodología pues hay indicios suficiente de que fracase la ejecución del proyecto.

A continuación se detallan los cuadros donde se definen los indicadores correspondientes a las preguntas esenciales y a las preguntas complementarias:

Preguntas Esenciales		
Individuo (que forman parte del Equipo de Trabajo)		
Indicador	**Peso**	**Valor**
Es auto organizado	2	
Tiene experiencia en el trabajo individual	2	
Se adapta sin inconvenientes al trabajo grupal	2	
Se centra únicamente en el cumplimiento de objetivos planteados (cumplimiento de hitos) en el Proyecto general	2	
Es autosuficiente y responsable en las tareas que les son asignadas	2	
Total	**10**	

Tabla 1. Indicadores esenciales agrupados por individuo

Preguntas Complementarias		
Gestión		
Indicador	**Peso**	**Valor**
Se mide el progreso del proyecto a través del seguimiento de funcionalidades, tareas y actividades	2	
Se realizan reuniones periódicas entre Project Manager y los Scrum Master utilizando herramientas de comunicación	2	
Se realizan reuniones cortas y frecuentas para conocer el estado de las tareas y actividades	2	
Se utilizar documentación/artefactos para mejorar la comunicación, coordinación y control del proyecto	2	
Existen Scrum Master On Site y Off Site	2	
Los Programadores, Tester Técnicos se sitúan Off Site	2	
La comunicación entre los Scrum Master In Site y Off Site es fluida	2	
Existe el intercambio de embajadores entre los sitios On Site y Off Site	2	
El Cliente, el Project Manager, Analistas Funciones y los Arquitectos se sitúan On Site.	2	
Total	**18**	
Metodología de desarrollo de software		
Indicador	**Peso**	**Valor**
Los equipos remotos integran e implementan internamente las prácticas de las metodologías XP y Scrum	2	
Los desarrolladores utilizan pruebas unitarias para comprobar	2	

su código		
Se utilizan integración continua	2	
Se realiza iteraciones cortas (de una o dos semanas), que vayan entregando funcionalidades incrementalmente	2	
Se utilizar frecuentemente construcciones para obtener feedback de las funcionalidades	2	
Total	**10**	
Uso de Herramientas		
Indicador	**Peso**	**Valor**
Se utilizan herramientas para la Gestión de proyectos distribuidos geográficamente	2	
Se utiliza una herramienta para producir construcciones automáticas a cada hora, hechas desde un repositorio común	2	
Se utiliza una herramienta para el control de versiones centralizado	2	
Se implementan herramientas formales e informales homologadas por todos los sitios remotos	2	
Se centraliza el acceso a las herramientas a través de: la Web, bases de datos replicadas, ambientes de desarrollo individuales y repositorios centrales	2	
Se utiliza una herramienta integral (Requerimientos, Diseño, Construcción, Pruebas e Implementación)	2	
Se utilizan simulaciones de ambientes	2	
Total	**14**	

Tabla 2. Indicadores complementarios agrupados por gestión, desarrollo y uso de herramientas

4. DESARROLLO DE LA METODOLOGIA PROPUESTA. CAPA 3

De acuerdo a todo lo expuesto anteriormente y para hacer frente al nuevo desafío que se está presentando en el último tiempo, desarrollar productos software por medio de equipos de trabajo distribuidos geográficamente, surge el desarrollo de la metodología denominada Capa 3. La metodología se centra en la elaboración de una estructura conformada por tres capas vinculadas, donde la superior estará formada por el segmento de Control o Gestión, que se encargará de administrar y coordinar el proyecto de software; luego, en la intermedia, se encuentran las metodologías de desarrollo de software que mejor se adapten al proyecto en ejecución y al desempeño de las tareas en ambientes virtuales, y la inferior estará conformada por el conjunto de herramientas colaborativas que soporten el desarrollo de las capas superiores y el proyecto de desarrollo de software en su totalidad.

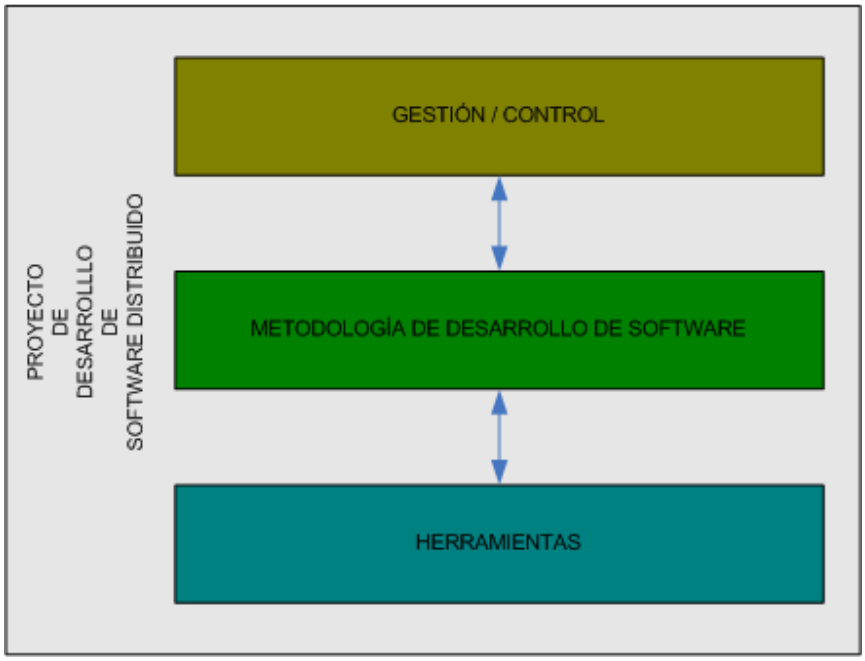

Figura 6. Representación gráfica de la metodología propuesta Capa 3.

4.1 Gestión

4.1.1. Introducción

Todo proyecto, bien de ingeniería o de cualquier otro campo, necesita de una adecuada gestión para obtener el control necesario del mismo a lo largo de su ciclo de vida.

Por ello, la gestión y el control son actividades que adquieren una importancia en sí mismas, de forma paralela al tema desarrollado por el propio proyecto. En concreto, para la realización de un proyecto de ingeniería nos encontramos con que no sólo se precisa el

saber técnico, sino también unos conocimientos y una técnica en la gestión de costos, del tiempo, de la calidad, de los riesgos, etc.

La formación clásica en Gestión de Proyectos se basa en un enfoque que asume un proceso altamente disciplinado dentro de una estructura que encaja dicho modelo en un esquema más o menos burocratizado y rígido. Esto es válido en entornos de trabajo homogéneos, grupos de desarrollo físicamente establecidos en un mismo ambiente y en donde todo el equipo de trabajo puede colaborar de una manera presencial. Sin embargo, la mayoría de los proyectos de desarrollo de software como los de sistemas (integración de hardware y software) en general exigen cada vez más un modelo ágil para la gestión de proyectos. Esta es la razón para que se ofrezca una formación en Gestión Ágil de Proyectos, dado que pueden obtenerse una serie de beneficios no conseguidos por las gestiones tradicionales de proyectos pues se puede planificar mejor la gestión centrado en los recursos que intervienen y en la distribución geográfica de los mismos.

La planificación se realiza sobre un análisis detallado del trabajo que se quiere realizar y su descomposición en tareas. Parte por tanto de un proyecto de obra, o de unos requisitos detallados de lo que se quiere hacer. Sobre esa información se desarrolla un plan adecuado a los recursos y tiempos disponibles; y durante la construcción se sigue de cerca la ejecución para detectar posibles desviaciones y tomar medidas para mantener el plan, o determinar qué cambios va a experimentar. Se trata por tanto de una gestión “predictiva”, que vaticina a través del plan inicial cuáles van a ser la secuencia de operaciones de todo el proyecto, su costo y tiempos.

Su principal objetivo es conseguir que el producto final se obtenga según lo “previsto”; y basa el éxito del proyecto en los tres puntos apuntados: agendas, costos y calidad.
La gestión de proyectos predictiva o clásica es una disciplina formal de gestión, basada en la planificación, ejecución y seguimiento a través de procesos sistemáticos y repetibles. Las organizaciones más conocidas por la investigación, y creación de comunidades profesionales para la gestión de proyectos son: PMI (Project Management Institute), International Project Management Association (IPMA) y Prince2.

Profundizando, podemos mencionar que a partir de 1970 es cuando la tecnología entra con fuerza como herramienta en la Gestión de Proyectos y se comienza a desarrollar software específico para dicha gestión, aglutinando las diferentes técnicas conocidas y simplificando procesos. Gracias a la tecnología aplicada a la Gestión de Proyectos surge una nueva línea de innovación e investigación en multitud de entornos: la simulación de proyectos [López Nieva A., 2009].

Incluso han existido ocasiones en las que la innovación ha tenido que reinventarse para reconducirse a sí misma. Un ejemplo conocido es la poco efectiva, en los años 90, de la

aplicación de métricas clásicas de otras disciplinas al desarrollo de software. Esto provocó el nacimiento de las metodologías ágiles.

Pero la innovación en la Gestión de Proyectos no se ha limitado únicamente a la mejora y perfección de sus técnicas y herramientas. El desvío de planificaciones y costos y la falta, en ocasiones, de calidad y utilidad de los entregables en los proyectos, que cada vez eran más complejos ya que requerían el trabajo conjunto y sincronizado de varias áreas o ingenierías, provocó que en los años 60 surgiera la necesidad de innovar y desarrollar nuevos métodos de organización y trabajo. Fue entonces cuando aparecieron diferentes organizaciones con el objetivo de desarrollar los conocimientos, metodologías y procesos y las prácticas necesarias para gestionar esos trabajos con las mejores garantías de previsibilidad y calidad de los resultados. Entre estas organizaciones encontramos el Project Management Institute (PMI) y el International Project Management Association (IPMA), ambos fundados en 1965. El PMI es una Asociación Internacional de Profesionales de Gestión de Proyectos, que cuenta con más de 200.000 miembros vigentes con presencia en más de 125 países por todo el mundo y más de 85.000 certificados PMP´s (Project Management Professional). Desde su fundación ha estado desarrollando y publicando estándares profesionales para la Gestión de Proyectos, los cuales se recogen actualmente en el PMBOK (Project Management Body of Knowledge), y que es la referencia de la norma ISO-10006 y su equivalente UNE-66916 de Octubre del 2003. En 1989 la Central Computer and Telecommunications Agency (CCTA) del Gobierno Británico desarrolló Prince2, un modelo de referencia para proyectos específicos de Tecnologías de la Información. A partir de una revisión llevada a cabo en 1996 se decidió ampliar su ámbito de validez, para cualquier tipo de proyecto [López Nieva A., 2009].

La innovación en la Gestión de Proyectos ha ido siempre también ligada a la innovación en Dirección Estratégica y Organización. Muestra de ello es la evolución que han experimentado tanto la propia Gestión de Proyectos como la figura del Director de Proyectos. Entre los años 1940 y 1970 podemos observar una Gestión de Proyectos tradicional. Se asume que las organizaciones viven en un mundo estático y predecible para desarrollar los proyectos y las decisiones se toman sin darle importancia a la satisfacción del cliente. El presupuesto, cronograma y la asignación de recursos se manejan a través de una serie de herramientas fijas y el ciclo de vida de proyecto consta solo de cuatro fases: Concepto, Planificación, Ejecución y Cierre. En cambio, en la Gestión de Proyectos moderna (desde 1980 hasta nuestros días) las cosas cambian, ya que se asume que vivimos en un mundo caótico y no predecible para desarrollar los proyectos y, al revés que la Gestión de Proyectos tradicional, para toma de decisiones se tiene en cuenta la opinión y satisfacción del cliente. Nacen nuevas técnicas y herramientas para gestionar los proyectos en los ámbitos de gestión de contratos, análisis de costos, análisis de riesgos, del entorno, negociación y gestión de cambios. Además, se incorporan dos fases adicionales al ciclo de vida tradicional: la fase de Mantenimiento y la fase de Operación [López Nieva A., 2009].

Pero una de las mayores innovaciones en la Gestión de Proyectos de los últimos años es, sin duda, la "humanización" del Gestor de Proyectos. Si analizamos el perfil del Gestor de Proyectos en la Gestión de Proyectos tradicional podemos observar que era un perfil altamente técnico y puramente implementador con conocimiento de la materia. Era disciplinario, con un alto sentido de la organización y el orden y perseverante en su trabajo, con un alto grado de compromiso. Por el contrario, estaba únicamente enfocado a resultados y no era un líder visible ni en el equipo de proyecto ni en la organización. Además, era inflexible en las decisiones [López Nieva A., 2009].

En la Gestión de Proyectos moderna el Gestor de Proyectos sigue manteniendo un alto grado de compromiso y teniendo conocimiento de la materia. Sigue siendo disciplinario, con un alto sentido de la organización y perseverante en su trabajo. Sin embargo sufre una profunda transformación, ya que se humaniza, porque lo más importante de un proyecto, al igual que en cualquier organización, son las personas, y son ellas las que tienen la llave para llevar el proyecto al éxito o al fracaso. Y el éxito o fracaso de esas personas será su éxito o fracaso. Piensa también en las personas del equipo de proyecto, es un líder visible, forma parte de equipo de proyecto y se preocupa por su gente. Es flexible y tolerante y desarrolla habilidades de comunicación efectiva, además de potenciar el empowerment de los miembros de su equipo y ejercer de mentor y coach de los mismos [López Nieva A., 2009].

En la actualidad el reto principal de la Gestión de Proyectos es la globalización, que provoca que la Gestión de Proyectos se expanda hacia entornos internacionales y por tanto, multiculturales. Por ello cada vez más se requieren conocimientos culturales específicos en idiomas y costumbres tanto de los países donde se desarrolla el proyecto como de los diferentes integrantes del equipo, para poder superar con éxito los retos con los cuales las empresas y los proyectos multiculturales se encuentran diariamente. La globalización es el ámbito donde más se está innovando actualmente gracias a que las organizaciones son conscientes de su importancia y se están adaptando a ella a través de formación multicultural. Otros retos actuales a los que se enfrenta la Gestión de Proyectos moderna son las exigencias para conseguir resultados más rápidos y con alta calidad en proyectos donde cada vez existe más un alto grado de incertidumbre y se multiplican los riesgos, el mantener un buen ambiente en el equipo de proyecto, adaptarse a las organizaciones planas y a las continuas reorganizaciones y fusiones y la excesiva dependencias de la tecnologías de la información.

Desde la pasada década, finales del siglo XX, está cada día más presente en las organizaciones la PMO[20], aunque todavía queda camino por recorrer. La Oficina de Gestión

20 (Project Management Office) u Oficina de Gestión de Proyectos.

de Proyectos nace formalmente producto del desarrollo de modernas herramientas y preceptos de gestión de proyectos para profesionalizar, automatizar y consolidar su manejo. La función principal de esta oficina es la de ser un elemento integrador entre el negocio y los diferentes proyectos de la empresa, consolidando iniciativas individuales en un solo portafolio; cuantificable, de fácil seguimiento y alineado a la estrategia de largo plazo de la organización [López Nieva A., 2009].

4.1.2. Gestión de proyectos de software distribuidos

En la International Conference on Complex Systems 2006 (ICCS2006) que se celebró del 25 al 30 de Junio de 2006 en Boston, Jeff Sutherland[21] afirmó que Scrum es la metodología de desarrollo capaz de lograr records de productividad y calidad, y que no sólo se puede aplicar en proyectos de pequeña envergadura. Dió a conocer el estudio "Adaptive Engineering of Large Software Projects with Distributed / Outsourced Teams[22]", en el que detalló cómo se ha trabajado y qué resultados se han obtenido en el desarrollo de un sistema Java de un millón de líneas de código, realizado por equipos distantes de más de 50 personas de las empresas SirsiDinix y StarSoft Development Laboratories, que han llevado a cabo el proyecto empleando Scrum + Extreme programming.

En la ponencia "Adaptive Engineering of Large Software Projects with Distributed / Outsourced Teams", Jeff Sutherland, Anton Viktorov y Jack Blout[23], afirmaron que con la aplicación de Scrum o Scrum + Extreme Programming lograron en el desarrollo de software muy buenos resultados de eficiencia y calidad, y que la afirmación de que se trata de una metodología adecuada para proyectos y equipos pequeños es un mito.

Los resultados de eficiencia han sido: un equipo de 57 personas en 14.5 meses (827 personas / mes) han desarrollado 671.688 líneas de código de gran calidad (con dos fases de refactorización incluidas).

Considerando las medias de productividad que suelen obtener los equipos que trabajan con ciclos de vida en cascada o secuenciales, Scrum multiplica la eficiencia por 10, ya que 54 personas/mes habrían obtenido los resultados que según afirma Mike Cohn en User Stories Applied: For Agile Software Development, obtiene el desarrollo en cascada con 540 personas/mes.

21 Creador de Scrum, metodología para la gestión y control de proyectos de desarrollo de software.
22 Ingeniería adaptativa de grandes proyectos de software con equipos distribuidos.
23 Anton Viktorov de la empresa StarSoft Development Laboratories – Jack Blout de la empresa SirsiDinix.

	SCRUM	Waterfall	SirsiDynix
Person Months	54	540	827
Lines of Java	50,803	54000	671,688
Function Points	959	900	12673
FP per dev/month	17.8	2.0	15.3
FP per dev/month (industry average)	12.5	12.5	3

Figura 7. Tabla comparativa de productividad [Sutherland, J., 2006]

4.1.3. Gestión de proyectos ágiles vs. Gestión de proyectos tradicional

Hay dos formas de gestionar proyectos: una predictiva o clásica y otra adaptativa o ágil. La gestión predictiva parte de un plan detallado. Sabe exactamente qué es lo que va a hacer y conoce fechas y costos. Durante el desarrollo gestiona riesgos, evalúa el impacto que cada modificación supone sobre el plan inicial, toma decisiones frente a los imprevistos para seguir su cumplimiento; y si no queda más remedio, para replantearlo.

La gestión adaptable parte desde una visión general del objetivo y va dando pequeños pasos hacia él a través de un ciclo de construcción incremental, y de forma evolutiva contrasta y va descubriendo el detalle que resulta más adecuado para el producto con los usuarios y demás participantes, que pueden "tocar" o usar las partes construidas. Hay un modelo que quiere conseguir previsibilidad: construir lo previsto, en el tiempo previsto y con el costo previsto. Hay un modelo que quiere conseguir valor competitivo en entornos rápidos de la economía actual: innovación y agilidad. Hay gestores que sólo trabajan de forma predictiva por la creencia fundamentalista de que es "la buena", y descalifican a los enfoques ágiles o adaptativos. Hay gestores que sólo trabajan de forma adaptativa por la creencia fundamentalista de que es "la buena", y descalifican a los enfoques clásicos o predictivos. El viaje de negocios de un ejecutivo necesita ajuste y previsión de agendas. Se debe gestionar con un patrón predictivo. Un viaje de turismo innovador para descubrir rutas poco frecuentes, capaz de cambiar rápidamente si empeora el tiempo en la zona, o sacar ventaja de la pérdida de un avión se debe gestionar con un patrón adaptable.

4.1.4. Gestión de proyecto sugerido

Una combinación en términos de complemento entre gestión ágil, (en el presente trabajo se toma a Scrum como metodología ágil de gestión de proyectos de desarrollo de software) y gestión tradicional basado en procesos y en la utilización de algunos artefactos de gestión

tradicional. Para la gestión de proyectos de desarrollo de software distribuidos geográficamente, se utiliza la combinación de artefactos de gestión ágil con artefactos de la gestión tradicional. La gestión es medida por el cumplimiento de los objetivos que se han establecido para cada equipo de desarrollo y el control de cumplimiento se va realizando por la finalización de las tareas que se fueron asignadas a cada equipo de desarrollo distribuido geográficamente. La gestión y control se hace en períodos de tiempos cortos a fin de detectar en etapas tempranas un posible desvío de tiempo en el proyecto. Si bien en el presente trabajo se deja de lado la gestión de riesgo, dejándola como un elemento a ser evaluado en lineamientos futuros, de todas maneras, es importante mencionar, que aquellos elementos con alto riesgo obtengan prioridad en el proceso de desarrollo y sean abordados en etapas tempranas del mismo. Para ello, se crean y mantienen listas identificando los riesgos desde etapas iniciales del proyecto. Especialmente relevante en este sentido es el desarrollo de prototipos ejecutables durante la base de elaboración del producto, donde se demuestre la validez de la arquitectura para los requisitos clave del producto y que determinan los riesgos técnicos.

Los equipos de desarrollo son autoorganizados. Esto significa que el plan se desarrolla en base a un objetivo o conjunto de objetivos definidos y no a las tareas concretas que inicialmente pueden haber sido previstas. El equipo elige el orden más adecuado en reuniones periódicas donde se informa de lo avanzado, de las dificultades encontradas y se determinan las acciones a realizar durante el día para cumplir con los objetivos del proyecto.

Las teorías de producción basadas en procesos funcionan bien en entornos de producción industrial, y de ahí los tomaron los pioneros de la ingeniería del software. Estos modelos trabajan sobre la premisa de que la calidad y la eficiencia de la producción se deben sobre todo a los procesos empleados.

De esta mixtura podemos decir que personas y procesos no son elementos simples, sino combinaciones “ejecución-talento” en el primer caso, y de “proceso-rutina” en el segundo; y con este criterio base flexibiliza los procedimientos y las estrategias de gestión, adoptando el grado más adecuado entre agilidad y disciplina para cada sub-sistema de la organización, tras analizar cuanto tiene el factor “personas” de ejecución y cuánto de trabajo; y qué grado de proceso y de rutina tiene el factor “procedimiento”.

4.2 Metodología de desarrollo de software

4.2.1. Introducción

El segmento o capa de Metodología explica de una manera abstracta y de alto nivel conceptual que características se debería tener en cuenta a la hora de implementar una o más metodologías (en combinación) de desarrollo de software para llevar a cabo un proyecto de elaboración de un producto software en ambientes heterogéneos. Esta capa actúa como un modelo conceptual en donde mediante la experiencia y la utilización de ciertos métodos dio lugar a un conjunto de directivas que se sugieren al momento de instaurar una metodología de desarrollo de software en ambientes de trabajo dispersos geográficamente.

Para comenzar podemos mencionar que existen muchos beneficios bien conocidos del desarrollo de software en ambientes virtuales distribuidos geográficamente, tales como un costo reducido de los productos debido principalmente a los menores costos laborales que se encuentran en países en desarrollo, como así también una mejora en la calidad general, principalmente debido a que las empresas que cada vez más se apegan a modelos CMM/CMMI a niveles 3 y superiores. Estos mismos proveedores están listos para utilizar sus procesos de desarrollo de software maduros al mismo tiempo que entienden la importancia de "Time-to-Market" adecuado.

Por otra parte la revolución de los métodos ágiles producida durante los últimos años ha traído una serie de avances en la disciplina de desarrollo de software; mejor calidad general y eficiencia basada en ciclos reducidos para la obtención de beneficios y organizaciones que logran cumplir con las necesidades de sus clientes en un entorno cambiante con sólo documentación básica y necesaria sumada a una gran interacción con sus usuarios. Como ventajas adicionales, el modelo ágil mejora la orientación al negocio o la respuesta adecuada al cambio, el cual lo hace muy eficiente al momento de mitigar riesgos mientras se mantiene un grupo de desarrollo motivado.

La combinación del desarrollo de software en ambientes virtuales distribuidos geográficamente y el modelo ágil es el “Santo Grial” hoy en día. Las ganancias exponenciales obtenidas a raíz de la mezcla de procesos ágiles de desarrollo con Offshore Outsourcing[24] bien valen el esfuerzo de su implementación.

Entonces, para entender bien el concepto de la Metodología, comencemos por recordar como surgió la denominación de “Métodos Ágiles”, piedra angular en el que se apoya el

24 Offshoring o outsourcing internacional es la subcontratación de procesos de negocios de un país a otro, usualmente en busca de costos más bajos o mano de obra.

presente trabajo (también sin olvidar los Métodos Formales o Tradicionales que sirven de apoyo en parte).

En marzo de 2001, 17 críticos de los modelos de mejora basados en procesos, convocados por Kent Beck, que había publicado un par de años antes el libro "Extreme Programming Explained" en el que exponía una nueva metodología denominada Extreme Programming, se reunieron en Salt Lake City para discutir sobre el desarrollo de software.
En la reunión se acuñó el término “Métodos Ágiles” para definir a los que estaban surgiendo como alternativa a las metodologías formales, (CMM-SW, PMI, SPICE) a las que consideraban excesivamente “pesadas” y rígidas por su carácter normativo y fuerte dependencia de planificaciones detalladas, previas al desarrollo.
Los integrantes de la reunión resumieron en cuatro postulados lo que ha quedado denominado como “Manifiesto Ágil”, que son los principios sobre los que se basan estos métodos.
Hasta 2005, entre los defensores de los modelos de procesos y los de modelos ágiles han sido frecuentes las posturas radicales, quizá más ocupadas en descalificar al otro que en estudiar sus métodos y conocerlos para mejorar los propios.

Si bien es importante destacar la diferencia de implementar una metodología ágil por sobre una metodología denominada formal/tradicional, es menester saber que en la presente tesis busca lograr un punto de equilibrio entre las dos “visiones” metodológicas, pues hay procesos y funciones muy destacables en las denominadas metodologías formales/tradicionales para llevar a cabo un desarrollo de software, como también la existencia de formalismos importantes en instaurar una metodología ágil sobre todo en ambientes de desarrollo dispersos geográficamente, como ser la orientación de las personas por sobre los procesos, pero sin un control formal de los procesos, teniendo en cuenta la distancias que separan a los integrantes de un equipo de desarrollo, resultaría muy complicado llevar a cabo un proyecto de desarrollo de software bajo un ambiente heterogéneo.

4.2.2. Manifiesto Ágil

A continuación se mencionan los manifiestos Ágil y cómo se relacionan con la metodología que se explica en la presente tesis a fin de dar sustento a la misma:

Valoramos más a los individuos y su interacción que a los procesos y las herramientas.
Este es posiblemente el principio más importante del manifiesto y el que mejor se aplica a la metodología propuesta en la tesis.

Por supuesto que los procesos ayudan al trabajo. Son una guía de operación. Las herramientas mejoran la eficiencia, pero en trabajos que requieren conocimiento tácito, sin personas con conocimiento técnico y actitud adecuada, no producen resultados. En

ambientes dispersos geográficamente es importante que cada individuo que participa en el desarrollo de un proyecto de software, ya sea concentrado en una célula de trabajo (equipo de trabajo remoto) o teletrabajando individualmente, tenga y lleve un control sobre las tareas que le fueron asignadas a fin de cumplir con los objetivos que se han planteado en el plan de proyectos.

Las empresas suelen predicar muy alto que sus empleados son lo más importante, pero la realidad es que en los años 90 la teoría de producción basada en procesos, la re-ingeniería de procesos ha dado a éstos más relevancia de la que pueden tener en tareas que deben gran parte de su valor al conocimiento y al talento de las personas que las realizan.

Los procesos deben ser una ayuda y un soporte para guiar el trabajo. Deben adaptarse a la organización, a los equipos y a las personas; y no al revés. La defensa a ultranza de los procesos lleva a postular que con ellos se pueden conseguir resultados extraordinarios con personas mediocres, y lo cierto es que este principio es peligroso cuando los trabajos necesitan creatividad e innovación.

Valoramos más el software que funciona que la documentación exhaustiva.
Poder ver anticipadamente como se comportan las funcionalidades que se esperan sobre prototipos o sobre partes ya elaboradas del sistema final ofrece un "feedback" muy estimulante y enriquecedor que genera ideas y posibilidades imposibles de concebir en un primer momento, y difícilmente se podrían incluir al redactar un documento de requisitos detallados antes de comenzar el proyecto.

El manifiesto no afirma que no hagan falta. Los documentos son soporte de documentación, permiten la transferencia del conocimiento, registran información histórica, y en muchas cuestiones legales o normativas son obligatorios, pero se resalta que son menos importantes que los productos que funcionan. Menos trascendentales para aportar valor al producto.

En proyectos de desarrollo de software en ambientes heterogéneos donde la dispersión geográfica de los integrantes de un equipo de desarrollo de software es la cuestión central del motivo del presente trabajo, es necesario que esos grupos de profesionales entiendan a la perfección los requerimientos que surgen del análisis y su posterior diseño. Si consideramos que los analistas de requerimientos y los diseñadores se encuentran geográficamente donde se encuentra el cliente y el equipo de desarrollo se establece en otra ubicación geográfica (incluso hablando un idioma distinto a los primeros), es necesario y de vital importancia que este último grupo interprete los requerimientos del cliente tal cual lo entendieron los analistas y los diseñadores. Pero para que el diseñador haya interpretado los requerimientos del cliente obtenidos por el analista y más tarde sea interpretado por el equipo de programadores es de importancia reforzar la documentación concebida por medio de la ingeniería de requerimientos a través de la generación de prototipos que

establezcan una idea/concepto de la funcionalidad de parte o de todo el producto software a desarrollar. Por eso es muy importante la generación de valor que se logra con la comunicación directa entre las personas y a través de la interacción con los prototipos.

Valoramos más la colaboración con el cliente que la negociación contractual.
Las prácticas ágiles están especialmente indicadas para productos difíciles de definir con detalle en el principio, o que si se definieran así tendrían al final menos valor que si se van enriqueciendo con retro-información continua durante el desarrollo. También para los casos también en los que los requisitos van a ser muy inestables por la velocidad del entorno de negocio.

Para el desarrollo ágil el valor del resultado no es consecuencia de haber controlado una ejecución conforme a procesos, sino de haber sido implementado directamente sobre el producto.

Un contrato no aporta valor al producto. Es una formalidad que establece líneas divisorias entre responsabilidades, que fija los referentes para posibles disputas contractuales entre cliente y proveedor.

En el desarrollo ágil el cliente es un miembro más del equipo, que se integra y colabora en el grupo de trabajo. Los modelos de contrato por obra no encajan.

Ponemos en claro que el cliente es una pieza importante de todo el equipo de desarrollo pues debe participar muy activamente interactuado con los equipos de desarrollo validando o no los prototipos que se diseñan con la finalidad de interpretar los aspectos funcionales de la aplicación que se está desarrollando. Hay que recordar que esta metodología busca relacionar de la mejor manera equipos de trabajo en entornos disímiles, dispersados geográficamente pero organizados en su conjunto con el objetivo de desarrollar un producto software. Por eso es muy importante que también el cliente esté unido al equipo de trabajo, pues es él quién aceptará o no las interpretaciones iniciales de los aspectos funcionales del producto para luego poner en marcha la elaboración del mismo de una manera adecuada cumpliendo con las expectativas del cliente en cuanto a lo que desea obtener como producto final.

Valoramos más la respuesta al cambio que el seguimiento de un plan
Para un modelo de desarrollo que surge de entornos inestables, que tienen como factor inherente el cambio y la evolución rápida y continua, resulta mucho más valiosa la capacidad de respuesta que la de seguimiento y aseguramiento de planes pre-establecidos. Los principales valores de la gestión ágil son la anticipación y la adaptación; diferentes a los de la gestión de proyectos ortodoxa: planificación y control para evitar desviaciones sobre el plan.

Si bien el Control en este trabajo es un parte esencial de toda la metodología, también es cierto que los procesos de desarrollo de software, desde su inicio, diseño, elaboración e implementación al estar ejecutados por equipos de profesionales disgregados geográficamente, todo el proyecto tiene que estar preparado para cambios rápidos y violentos, siendo la causa una mala interpretación del equipo de desarrollo a los requerimientos volcados por los ingenieros de requerimientos quizás ubicados y trabajando en otro lugar físicamente.

4.2.3. La experiencia del modelo Open Source[25]

Después de todo el código abierto es un estilo de software, no tanto un proceso. Sin embargo hay una manera definida de hacer las cosas en la comunidad de código abierto, y mucho de su acercamiento es tan aplicable a los proyectos de código cerrado como a los de código abierto. En particular su proceso engrana equipos de trabajo físicamente distribuidos, lo que es importante porque la mayoría de los procesos adaptables exigen equipos locales.

La mayoría de los proyectos de código abierto tienen uno o más mantenedores. Un mantenedor es la única persona a la que se le permite integrar un cambio en el almacén de código fuente. Sin embargo otras personas pueden hacer cambios a la base del código. La diferencia importante es que estas otras personas necesitan enviar su cambio al mantenedor que entonces lo revisa y lo aplica a la base del código. Normalmente estos cambios son hechos en forma de archivos de parches que hacen este proceso más fácil. Así, el mantenedor es responsable de coordinar los parches y mantener la cohesión en el diseño del software.

Proyectos diferentes manejan el papel del mantenedor de diferentes maneras. Algunos tienen un mantenedor para el proyecto entero, algunos lo dividen en módulos y tiene un mantenedor por módulo, algunos rolan el mantenedor, algunos tienen múltiples mantenedores sobre el mismo código, otros tienen una combinación de estas ideas. La mayor parte de la gente de código abierto son de tiempo parcial, así que hay una duda en qué tan bien se coordina un equipo así para un proyecto de tiempo completo.

Un rasgo particular del desarrollo de código abierto es que la depuración es altamente paralelizable. Muchas personas pueden involucrarse en el depurado. Cuando encuentran un defecto pueden enviar el parche al mantenedor. Esto es un buen papel para los no mantenedores ya que la mayor parte del tiempo se gasta en encontrar bugs. También es bueno para gente sin mucha destreza en programación.

25 OS por su sigla en inglés de Open Source. En castellano, Código Abierto.

El proceso para el código abierto aun no se ha documentado bien. La referencia más famosa es el artículo de Eric Raymond [1998] "La Catedral y el Bazar", que aunque es una descripción excelente también es bastante informal. El libro de Karl Fogel [1999] sobre "Desarrollo de Código Abierto con CVS[26]", también contiene varios buenos capítulos sobre el proceso de código abierto que incluso serían interesantes para aquellos que no quieren usar el CVS. CVS es un sistema cliente-servidor que permite a los desarrolladores realizar el seguimiento de las diferentes versiones del código fuente de un proyecto. Se trata de una herramienta muy potente que facilita:

- Registrar todos los cambios efectuados sobre los archivos de un proyecto.
- Recuperar versiones anteriores del código de un proyecto.
- Conocer qué cambios se han efectuado sobre un archivo determinado, quién los ha realizado y cuándo.
- Gestionar los conflictos que pueden producirse en entornos en los que los desarrolladores se encuentran distribuidos geográficamente.

El CVS es especialmente útil cuándo más de una persona trabaja sobre un archivo específico. En tales situaciones, es posible que un desarrollador sobrescriba accidentalmente los cambios que ha realizado otro desarrollador. CVS resuelve este problema haciendo que cada desarrollador trabaje sobre su propia copia del código (lo que se denomina "workspace" o espacio de trabajo) y permitiendo posteriormente unir los cambios de todos los desarrolladores en un repositorio común.

4.2.4. Diferencias entre metodologías Ágiles y Tradicionales

El desarrollo de software no es una tarea fácil. Prueba de ello es que existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Por una parte tenemos aquellas propuestas más tradicionales que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos. Una posible mejora es incluir en los procesos de desarrollo más actividades, más artefactos y más restricciones, basándose en los puntos débiles detectados. Sin embargo, el resultado final sería un proceso de desarrollo más complejo que puede incluso limitar la propia habilidad del equipo para llevar a cabo el proyecto. Otra aproximación es centrarse en otras dimensiones, como por ejemplo el factor humano o el producto software. Esta es la filosofía de las metodologías ágiles, las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los

26 CVS son las siglas de Concurrent Versions System (Sistema de Versiones Concurrentes)

tiempos de desarrollo pero manteniendo una alta calidad. Las metodologías ágiles están revolucionando la manera de producir software, y a la vez generando un amplio debate entre sus seguidores y quienes por escepticismo o convencimiento no las ven como alternativa para las metodologías tradicionales.

En las dos últimas décadas las notaciones de modelado y posteriormente las herramientas pretendieron ser las "balas de plata" para el éxito en el desarrollo de software, sin embargo, las expectativas no fueron satisfechas. Esto se debe en gran parte a que otro importante elemento, la metodología de desarrollo, había sido postergado. De nada sirven buenas notaciones y herramientas si no se proveen directivas para su aplicación. Así, esta década ha comenzado con un creciente interés en metodologías de desarrollo. Hasta hace poco el proceso de desarrollo llevaba asociada un marcado énfasis en el control del proceso mediante una rigurosa definición de roles, actividades y artefactos, incluyendo modelado y documentación detallada. Este esquema "tradicional" para abordar el desarrollo de software ha demostrado ser efectivo y necesario en proyectos de gran tamaño (respecto a tiempo y recursos), donde por lo general se exige un alto grado de ceremonia en el proceso. Sin embargo, este enfoque no resulta ser el más adecuado para muchos de los proyectos actuales donde el entorno del sistema es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. Ante las dificultades para utilizar metodologías tradicionales con estas restricciones de tiempo y flexibilidad, muchos equipos de desarrollo se resignan a prescindir del “buen hacer” de la ingeniería del software, asumiendo el riesgo que ello conlleva. En este escenario, las metodologías ágiles emergen como una posible respuesta para llenar ese vacío metodológico. Por estar especialmente orientadas para proyectos pequeños, las metodologías ágiles constituyen una solución a medida para ese entorno, aportando una elevada simplificación que a pesar de ello no renuncia a las prácticas esenciales para asegurar la calidad del producto [Letelier, P. y Penadés, M. C., 2008].

Las metodologías tradicionales imponen una disciplina de trabajo sobre el proceso de desarrollo de software, con el objetivo de asegurar que el software que se obtenga satisfaga los requerimientos del usuario y reúna estándares aceptables de calidad. El trabajo de planificación es riguroso, aún cuando en la práctica muchas veces estas planificaciones no se respetan. En contraposición, las metodologías ágiles aportan nuevos métodos de trabajo que apuestan por una cantidad apropiada de procesos. Es decir, no se desgastan con una excesiva cantidad de cuestiones administrativas (planificación, control, documentación) ni tampoco defienden la postura extremista de total falta de proceso. Ya que se tiene conciencia de que se producirán cambios, lo que se pretende es reducir el costo de rehacer el trabajo.

Se identifican como principales diferencias entre ambos enfoques las siguientes:

- Las metodologías ágiles son adaptativas más que predictivas. Una metodología tradicional potencia la planificación detallada y de largo alcance de prácticamente todo el desarrollo de software (ejemplo Modelo Cascada). En contraste, las metodologías ágiles proponen procesos que se adaptan y progresan con el cambio, llegando incluso hasta el punto de cambiar ellos mismos.

- Las metodologías ágiles están orientadas más a los desarrolladores que a los procesos de desarrollo. Intentan entonces trabajar con la naturaleza de las personas (desarrolladores y usuarios) asignadas a un proyecto. Permitiendo que las actividades de desarrollo de software se conviertan en una actividad de colaboración grata e interesante.

Se presentan algunas limitaciones a la aplicación de los procesos ágiles en algunos tipos de proyectos. Los agilistas y sus críticos focalizan la discusión en torno a temas como la documentación y codificación. Por un lado se sostiene que el código es el único entregable que realmente importa, desplazando el rol del análisis y diseño en la creación de software. Los críticos precisan que el énfasis en el código puede conducir a la pérdida de la memoria corporativa o conocimiento organizacional, porque hay poca documentación y modelos para apoyar la creación y evolución de sistemas complejos.

4.2.5. Matriz comparativa de las dos metodologías

A continuación se grafica mediante una matriz los aspectos generales relevantes de cada metodología. El fin de esta matriz es interpretar las mejores cualidades de cada orientación con el objetivo de incorporarlas en la metodología de desarrollo de software en entornos heterogéneos propuesto en el presente trabajo.

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	Grupos grandes y posiblemente distribuidos

trabajando en el mismo sitio	
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 3. Matriz comparativa entre metodologías ágiles y tradicionales [Letelier y Penadés, 2008]

A continuación se detallan las características principales de las dos metodologías, agrupadas por Aplicación, Gerenciamiento de proyecto, Aspectos técnicos, Personal y Cultura. Con estos grupos de características nos permite tener una visión clara de las acciones y procedimientos que se llevan a cabo en cada una de las orientaciones. Nos permite identificar sus cualidades principales estableciendo un criterio para adecuarlo a la metodología de desarrollo de software en ambientes virtuales del presente trabajo.

Característica	Ágil	Tradicional
Aplicación		
Objetivo primario	Obtener valor rápida y continuamente; responder al cambio	Alta seguridad, predecible, repetible, optimizable
Tamaño	Grupo y proyecto pequeño	Grupo y proyecto grande
Entorno	Turbulentos, de alto cambio, foco en el proyecto	Estables, pocos cambios, foco en proyecto y organización, tomar en cuenta el entorno operativo existente
Gerenciamiento del proyecto		
Relación con clientes	Clientes en el lugar; focalizados en priorizar requerimientos	Interacción con clientes según se requiera; focalizado en contratos
Planificación y control	Planes internalizados; control cualitativo	Planes documentados; control cuantitativo
Comunicación	Conocimiento tácito e interpersonal	Conocimiento explícito y documentado
Aspectos técnicos		
Requerimientos	Historias informales y casos de pruebas priorizados; con cambios no predecibles	Especificaciones formales y completas bajo control de cambios
Desarrollo	Diseño simple; iteraciones cortas; se asume que el costo del cambio es bajo y	Arquitectura; iteraciones mayores; se asume que el costo del cambio es creciente

	constante	
Pruebas	Casos de pruebas ejecutables definen requerimientos	Plan y procedimientos de prueba
Personal		
Clientes	Dedicados y en el lugar; características CRACK (collaborative, representative, authorized, committed, knowledgeable)	Características CRACK y deben tener alta participación en algunos momentos, pero no necesariamente en toda la duración del proyecto
Desarrolladores	Alto porcentaje de recursos seniors, el resto semi-seniors	Alto porcentaje de recursos senior al inicio, luego los perfiles distribuidos
Cultura	Empowerment a través de autonomía	Empowerment a través de políticas y procedimientos

Tabla 4. Características principales de las metodologías ágiles y tradicionales [Gabardini, J., Campos, L., 2004]

4.2.6. Flexibilidad ante todo

Existen dificultades típicas de los proyectos clásicos, dificultades que han llevado a criticar cada vez más las metodologías clásicas -denominadas "de cascada"- debido al encadenamiento sucesivo de etapas bien delimitadas (análisis de las necesidades, diseño, especificaciones, etc.). Normalmente se les atribuyen dos grandes inconvenientes: por un lado, la dificultad para adaptarlas a especificaciones variables que, a menudo, son inevitables en los proyectos de larga duración y, por otro lado, la pesadilla que implica la fase de integración, en la que es necesario sincronizar todos los módulos desarrollados por equipos independientes de desarrollo. Lógicamente, estos dos aspectos se multiplican en los proyectos de desarrollo de software con equipos distribuidos geográficamente, en los que la distancia complica la incorporación de las especificaciones y la integración final.

El análisis de estas metodologías, con el tiempo, ha dado lugar a un nuevo tipo, las denominadas metodologías ágiles. Su característica principal es que utilizan ciclos de iteración cortos a lo largo de todo el proyecto y encadenan rápidamente todas sus etapas, desde el planteamiento inicial hasta las pruebas funcionales del módulo desarrollado. La duración de los ciclos puede variar, pero suele ser de unos quince días. Esto conlleva claras ventajas: por un lado, permite adaptarse rápidamente a los cambios y variaciones en las especificaciones, sobre todo los que se derivan del análisis de los resultados realizado por el cliente al final del ciclo; y por otro lado, se simplifica notablemente la integración, ya que se va realizando a medida que avanzan los ciclos. Así pues, una de las primeras recomendaciones que podría hacerse a la hora de trabajar en un proyecto de desarrollo de software con equipos distribuidos geográficamente es utilizar este tipo de metodología para simplificar toda una serie de problemas que suelen surgir en este contexto.

4.2.7. Metodologías Ágiles adaptadas a entornos de trabajo distribuidos

A continuación se mencionarán las principales características de la metodología propuesta:

- Es una metodología orientada a las personas y como la gente organiza los procesos dentro de la metodología de desarrollo adoptada, por lo que no es una metodología orientada totalmente a los procesos. Siguiendo dicho razonamiento es por lo que se resaltan las metodologías ágiles a fin de implementar la metodología propuesta en el presente trabajo pero logrando un equilibrio con algunos aspecto importantes de las metodologías denominadas formales, tradicionales o "duras". Si bien las metodologías ágiles son importantes pues se caracterizan en los valores personales mas allá de los procesos que sustentan cualquier desarrollo de software, es imperante también no dejar de lado las metodologías "duras", pues son esas metodologías y su forma de llevar a cabo los procesos de desarrollo las que permiten realizar un mejor seguimiento y control de los proyectos.

- Por otro lado los desarrollos en ambientes heterogéneos, donde la producción (procesos de desarrollo de software) se realiza en una lejana localidad llevada adelante por expertos/especializados programadores, a veces resulta dificultoso el entendimiento de las necesidades del cliente y de donde se realizo la ingeniera de requerimientos llevada acabo por el equipo de analistas, lejos de percibir la necesidad concreta del cliente. Esto conlleva a abordar una metodología dinámica, clara, focalizada, que permita al equipo de desarrollo cambiar la codificación, si así se presenta, por consecuencia del poco entendimiento de los requerimientos incluso cuando el proyecto se encuentra en estado de ejecución.

- Es importante que el cliente sea parte del equipo de desarrollo del producto software, pues es una pieza fundamental a la hora de interpretar correctamente y contrastar los requerimientos funcionales obtenidos en la etapa de ingeniería de requerimientos; para ello es de importancia el prototipado de las funcionalidades de parte o de todo el producto software a desarrollar a fin de estar seguro que lo solicitado por el cliente es lo que él espera de la aplicación.

- Es importante utilizar un modelo "integrado scrum", donde se integran todos los equipos de desarrollos que se encuentran distribuidos en diferentes puntos geográficos, con esto se asegura la comunicación, la cooperación entre los equipos de desarrollo más allá de la distancia, ya que la comunicación entre los equipos de trabajo es respaldado por herramientas y tecnología a fin de establecer una relación entre las partes que intervienen en el proceso de desarrollo del producto software.

4.2.8. Una combinación perfecta

Uno puede pensar que cualquier proceso que sea usado debería ser ágil o ligero para reducir al mínimo los gastos de comunicación y permitir una fácil medición del progreso basada en un producto más que documentos terminados. Probablemente es necesario que el modelo de proceso sí sea adaptable. Esos criterios sugieren que un modelo ágil de proceso es el indicado.

Pero la agilidad parece poco lógica en un entorno de desarrollo distribuido porque ésta requiere de una estrecha colaboración. Por lo tanto, la localización de los equipos debe ser la ideal. Asimismo, más, no menos, la documentación es fundamental para comunicar información inequívocamente a equipos remotos. Por lo tanto, necesitamos un modelo de proceso que goce de algunas características de los modelos tradicionales y algunas características de los modelos ágiles.

Por lo antes mencionado, a la hora de establecer un entorno distribuido de desarrollo de software, hay que considerar los siguientes aspectos:

- Incrementar la percepción sobre lo que ocurre en el proyecto, la compañía y los sitios remotos. Esta medida reduce la posibilidad de malos entendidos, conflictos y fallas de coordinación.

- Realizar una división de trabajo eficiente. Una repartición de trabajo basada en las capacidades, aumenta la probabilidad de éxito cuando hay fuertes relaciones laborales y experiencia. En cambio, si dichas características son escasas se recomienda una división de trabajo basada en las funcionalidades. Considerar el impacto que produce cambiar las asignaciones de tareas establecidas, porque influye negativamente en la motivación de los integrantes del equipo, pudiendo afectar el éxito del proyecto.

- Facilitar la flexibilidad laboral, porque permite la superposición de los diferentes horarios laborales incrementando la comunicación en tiempo real.

- Facilitar el seguimiento de errores de programación y el desarrollo de tareas. Esta práctica se considera críticas para lograr proyectos exitosos.

- Flexibilizar la gestión del proyecto y establecer objetivos claros, realizando una planificación semanal flexible que permita la adaptación dinámica de las tareas diarias.

- Establecer comunicación sistemática, fijando reglas de estilo y frecuencia, reduciendo el riesgo de malos entendidos y conflictos, además relacionándose positivamente con una comunicación efectiva y la satisfacción personal de los integrantes.

4.2.9. Organización y gestión: equipos distribuidos geográficamente motivados

Un primer punto que debe analizarse es la distribución entre los equipos de desarrollos dispersos geográficamente y los equipos locales. Evidentemente, no resulta fácil establecer una norma general, aunque se estima que una buena forma de repartirlos es un 30% local (pudiendo incluirse aquí los colaboradores del proveedor de servicios informáticos offshore) y un 70% offshore. Evidentemente, esta distribución puede ajustarse en función del contexto en el que se realiza el proyecto. Por ejemplo, si se quiere implementar un paquete de gestión integrada, en el que el aspecto funcional tiene gran importancia, la proporción debería ser de 40/60 [Cerf, S., 2006]. Y si hilamos aún más fino, otro punto que merece nuestra atención es la distribución de las funciones locales y offshore.

En este caso se recomienda no relegar los equipos offshore meramente a las funciones de desarrollo, sino que se aconseja implicarlos en otros eslabones de la cadena, por ejemplo, haciéndolos partícipes tanto en el diseño como en la integración. Esto garantiza una comunicación más fluida, una mayor motivación y, además, una alta productividad, ya que se reduce su dependencia de los equipos locales.

La productividad es, precisamente, uno de los aspectos que hay que vigilar más de cerca. No debemos olvidar que las relaciones a distancia suelen ser difíciles y, por tanto, resulta crucial asegurar un intercambio constante con los equipos offshore a todos los niveles de la organización y, no menos importante, valorar el trabajo bien hecho.

Cuando la distancia se interpone en la gestión de los equipos (como es el caso de los desarrolladores que trabajan en un mismo módulo), la experiencia nos dice que se necesita disponer de un gestor sobre el terreno para evitar los problemas que conlleva la gestión a distancia. Así pues, no hay que dudar a la hora de utilizar más gestores si queremos mejorar la eficacia.

Por último, no debemos olvidar la figura del coordinador offshore, una responsabilidad que debe confiarse a personas con una experiencia contrastada en este tipo de proyectos.

4.2.10. Caso de éxito de desarrollo de software distribuido

Sutherland [2007] señala un conjunto de prácticas denominadas “Distributed Scrum”, según este autor éstas serían las responsables del éxito logrado en el proyecto “Horizon

8.0" elaborado por la empresa SirsiDynix[27] (56 desarrolladores distribuidos entre EEUU, Canadá y Rusia). Las prácticas realizadas para alcanzar el éxito fueron las siguientes:

- Reuniones diarias entre el jefe de proyecto y los miembros de los sub-equipos remotos (mediante teleconferencia). Estas reuniones se denominaron reuniones scrum porque estaban basadas en las reuniones propuestas por la metodología Scrum. Las reuniones tuvieron cuatro características principales: trataban un conjunto de preguntas estándares, en un periodo de tiempo reducido, a través de un protocolo específico e involucraban a todos los miembros que estaban ubicados remotamente. El protocolo específico consistió en enviar las respuestas por correo electrónico antes del inicio de cada reunión, lo que ayudó a derribar barreras culturales y diferencias de estilo. Mientras que las preguntas estándares fueron:
    - ¿Qué se logró ayer?
    - ¿Qué se realizará hoy?
    - ¿Qué impedimentos han surgido para lograr las metas?

- Cada sub-equipo realizó reuniones diarias, al comienzo de la jornada laboral, para coordinar las actividades periódicas.
- Los jefes de los sub-equipo tuvieron cada semana una reunión Scrum en un día determinado.
- El cliente, el jefe de proyecto y los arquitectos se situaron en una misma localidad.
- Los arquitectos tuvieron, una vez a la semana, reuniones Scrum presenciales.
- Para captar los nuevos requisitos y ajustar las funciones elaboradas el jefe del proyecto y los analistas tuvieron reuniones diarias con el cliente.
- Todos los desarrolladores tuvieron las mismas condiciones.
- Los equipos remotos integraron e implementaron internamente las prácticas de las metodologías XP y Scrum, los más destacadas fueron:
    - Programación en parejas (Pair programming)
    - Refactorización (Refactoring)
    - Construcción continua (Continuous Integration)

- Para desarrollar las funcionalidades, cada vez que se pudo, se reutilizó código funcional para adaptarlo a las necesidades del cliente.
- Todos los miembros del equipo, proveniente de cualquier sitio, pudieron trabajar sobre todas las tareas del proyecto.
- Los requisitos fueron captados a través de historias de uso, algunos ampliamente detalladas y otras no. Las dudas surgidas se despejaron a través de reuniones scrum, mensajería instantánea y/o a través de e-mails.

27 Empresa de EEUU que desarrolla software. Su principal centro se encuentra en Huntsville, Alabama, y cuenta con 400 empleados distribuidos en todo el mundo.

- Cada dos semanas se realizaron reuniones de planificación, donde se solicitaron las historias de uso, posteriormente éstas se dividían en tareas y sub-tareas de desarrollo. Se denominó a los periodos entre cada reunión como Sprint, al final de cada uno de estos plazos debían estar implementadas las funcionalidades solicitadas, ya sea a modo de prototipo o de forma definitiva.
- Los desarrolladores utilizaron pruebas unitarias para comprobar su código; el cliente utilizó sondeos manuales; y los equipos de pruebas automáticas y manuales.
- Se utilizó una herramienta para producir construcciones automáticas a cada hora, hechas desde un repositorio común; después de cada construcción el sistema envió un e-mail a los desarrolladores informando sobre este proceso. Sin embargo, sólo se solicitaron construcciones estables al fin de cada Sprint.
- Se utilizó una herramienta especial para sustentar el repositorio y se proveyó de una efectiva ingeniería concurrente. Primero se utilizó la herramienta Open Source CVS.
- Se midió el progreso del proyecto a través del seguimiento de funcionalidades, tareas y actividades. Para esto se utilizó la herramienta de gestión de proyectos Jira. Esta herramienta le brindó a cada integrante una visión en tiempo real del estado de cada Sprint.

En cuanto a las prácticas propuestas, son el resultado de cuatro años de experiencia (desde comienzos de 2002 hasta comienzos de 2006) en proyectos desarrollados en la India, América del Norte y Europa [Fowler, 2006]. Enfocándose principalmente en utilizar prácticas ágiles en los proyectos distribuidos. Las prácticas propuestas son:

- Usar integración continua para evitar los problemas
- Intercambio de embajadores entre los sitios remotos
- Utilizar visitas presenciales para construir confianza
- No subestimar los cambios culturales
- Utilizar wikis para almacenar información común
- Utilizar Test Scripts o Prototipado para ayudar a entender los requisitos
- Utilizar frecuentemente construcciones para obtener feedback de las funcionalidades
- Realizar reuniones cortas y frecuentas para conocer el estado de las tareas y actividades
- Realizar iteraciones cortas (de una o dos semanas), que vayan entregando funcionalidades incrementalmente
- Utilizar una planificación de iteración adaptada a los sitios remotos, según la restricciones propias de cada sitio (por ejemplo a las diferencias horarias y culturales)
- Realizar reparación de defectos para dar a conocer el código
- Separar los equipos por funcionalidades y no por actividades

- Utilizar más documentación, que en los proyectos ágiles colocalizados, para mejorar la comunicación, coordinación y control
- Implementar múltiples formas de comunicación desde el inicio del proyecto

4.2.11. Modelo propuesto de distribución de equipos de trabajo

El modelo propuesto en el presente trabajo sigue el lineamiento sugerido por Jeff Sutherland [2007] como caso de éxito en el desarrollo de software con equipos de trabajo distribuidos. Si bien hay diferentes modelos para distribuir los equipos de trabajo, los cuales dependen de muchos factores, como la capacidad de cada integrante, la experiencia, el idioma, la cultura, el tipo de proyecto, etc., entonces podemos mencionar que el modelo de distribución de equipos de trabajo planteado en el actual trabajo tiene como principal las siguientes características de cada miembro que lo conforman:

- son auto organizados,
- tiene experiencia en el trabajo individual,
- se adapta sin inconvenientes al trabajo grupal,
- se centra únicamente en el cumplimiento de objetivos planteados (cumplimiento de hitos) en el Proyecto general,
- es autosuficiente y responsable en las tareas que les fueron asignadas.

Los equipos globales difieren de los equipos múltiples establecidos en un mismo lugar, o "coestablecidos". Los equipos coestablecidos instintivamente iniciarán un proceso de colaboración informal. Este es un comportamiento humano natural que gradualmente lleva a un mayor nivel de confianza y comprensión mutua. Cuando los miembros del equipo han establecido una relación de confianza, comenzarán a formular preguntas unos a otros, y a ofrecer información de manera proactiva. Además, podrán estimar las capacidades de los demás de una mejor manera, lo cual genera una división del trabajo más efectiva.

Teniendo como base lo antes descrito, se diseño un esquema de colaboración desde dos ambientes de trabajo, el denominado On Site[28] y el Off Site[29]. El ambiente de trabajo On Site está ligado exclusivamente desde donde se imparten los requerimientos del Proyecto, es decir, donde se encuentra establecido físicamente el Cliente. Luego, junto al Cliente, se encuentran el resto de lo responsables unidos al negocio, los stakeholders, el Scrum Master, el Project Manager y los Analistas Funcionales. En definitiva, se encuentran los actores de negocio, los responsables de transmitir los requerimientos y los de analizar los mismos junto con los coordinadores principales del Proyecto.

Mientras que el ambiente de trabajo Off Site, está compuesto por todos aquellos actores cuyos roles son operativos, están compuesto por los diseñadores, programadores,

28 On Site: lugar físico donde se encuentra el cliente y los responsables del negocio
29 Off Site: lugar físico fuera del ámbito del negocio y del cliente

integradores y testers. Mientras que los arquitectos pueden estar On Site u Off Site, dependiendo de las características del Proyecto de desarrollo de software.

A continuación se esquematiza los actores que intervienen en cada uno de los roles definidos en el modelo propuesto:

Actores On Site	Actores Off Site
Cliente	Scrum Masters
Stakeholders	Programadores
Project Manager	Testers Técnicos
Scrum Master	Embajadores
Analistas Funcionales	
Arquitecto/Diseñador	
Testers Funcionales	
Embajadores	

Tabla 5. Actores que intervienen en el modelo propuesto

A continuación se grafica las etapas del ciclo de vida de desarrollo de software propuesto junto con los ámbitos de trabajo propuestos:

Etapas de Ciclo de Vida de Desarrollo de Software	Localización
Modelamiento del Negocio	On site
Requerimientos	On Site
Diseño y Arquitectura	On Site / Off Site
Construcción	Off Site
Pruebas	Off Site / On site
Implementación	On site

Tabla 6. Etapas del ciclo de vida de desarrollo de software propuesto

El proyecto de desarrollo de software distribuido es coordinado por un Project Manager. Su función es controlar y guiar a todos los equipos de desarrollo y en definitiva a todos los actores que intervienen en el proceso de desarrollo de software. Tiene una visión macro del proyecto y es quién determina los avances del mismo. Está en comunicación permanente con el Scrum Master On Site como con los Scrum Master Off Site que pertenecen a los equipos de desarrolladores que se encuentran distribuidos geográficamente.

El Modelamiento del Negocio se realiza donde se encuentra físicamente el cliente y los stakeholders ya que ellos son los que sugieren el desarrollo de un Proyecto de software y son quienes determinan como debe estar alineado al negocio el producto software a desarrollar. Es importante que los actores estén en el mismo lugar físico ya que es importante la fluidez de las comunicaciones y la toma de decisiones preliminares para el desarrollo del Proyecto de software.

La ingeniería de Requerimientos, de la misma manera que la etapa anterior, debe realizarse donde se encuentra el Cliente quien es el que conoce el negocio y define los aspectos funcionales del producto software que se desarrollará. El Analista Funcional es el actor principal en esta etapa ya que modela los requerimientos suministrados por el Cliente para que luego, en la etapa de Diseño y Arquitectura, sea interpretado por el Arquitecto de software y de esta manera pueda diseñar la estructura de componentes que se desarrollará y/o se reutilizará en la construcción del producto software.

La Construcción del software es realizado por grupos de equipos de programadores distribuidos geográficamente, supervisados por un Scrum Master. La integración continua distribuida está centralizada en un único repositorio y es mantenida por cada equipo de desarrollo. Las pruebas técnicas son realizadas por un equipo de testers técnicos que pueden estar distribuidos geográficamente, mientras que el grupo de testers funcionales deben permanecer cerca del Cliente.

La implementación de las funcionalidades que se desarrollan y que son establecidas en cada Sprint, y que surgen de la integración continua, se llevan a cabo donde se encuentra el Cliente y los stakeholders ya que son los encargados de evaluar el producto software si cumple con los requerimientos establecidos.

Con el objetivo de lograr afianzar la confianza entre las partes On Site y Off Site es recomendable utilizar embajadores[30] ya que serán los responsables de mantener las relaciones de confianza y las comunicaciones aún más claras entre los equipos de desarrolladores remotos Off Site y los actores locales On Site.

A continuación se grafica el modelo sugerido de distribución del equipo de trabajo en el proceso de desarrollo de software distribuido:

30 Embajadores: son aquellas personas involucradas en un proyecto de desarrollo de software distribuido geográficamente. Se encargan de afianzar las comunicaciones y las relaciones entre las partes On Site y Off Site.

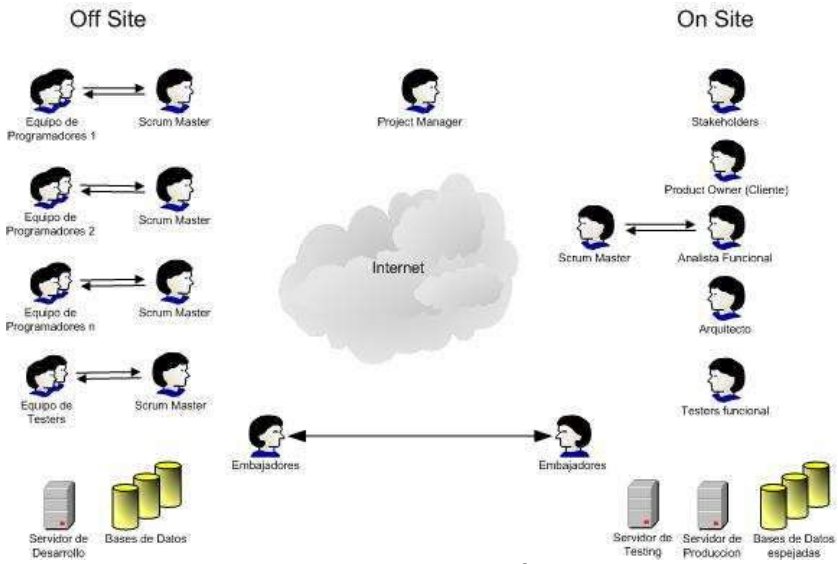

Figura 8. Modelo sugerido de distribución del equipo de trabajo

A continuación se detallará las principales características, funciones y tareas de cada involucrado en el Proyecto de desarrollo de software distribuido geográficamente definido por el modelo propuesto en el presente trabajo:

4.2.11.1. Actores involucrados en la localización On site

Stakeholders:

También llamados interesados o involucrados en un problema determinado que necesitan una solución óptima. Desde el punto de vista del desarrollo de sistemas, un stakeholders es aquella persona o entidad que esta interesada en la realización de un proyecto o tarea, auspiciando el mismo ya sea mediante su poder de decisión o de financiamiento.

En la gestión de proyectos, los involucrados o interesados son todas aquellas personas u organizaciones que afectan o son afectadas por el proyecto, ya sea de forma positiva o negativa. Una buena planificación de proyectos debe involucrar la identificación y clasificación de los interesados, así como el estudio y la determinación de sus necesidades y expectativas.

Cliente/Product Owner:

Cliente fundamental del sistema. El que desea o propone el software. El propietario del producto. El 'input' principal para poner en marcha el nuevo proyecto. Es fundamental que la persona que desempeña este rol se implique al máximo en el proyecto, para que éste tenga garantías de éxito.

Analista Funcional:
El Analista Funcional es la persona o grupo de personas que se tienen que encargar de entender, interpretar y traducir lo que el Cliente demanda, sentando las bases de los posteriores procesos de diseño y construcción del sistema de información.

Arquitecto/Diseñador:
La función principal del Arquitecto/Diseñador de software en el modelo sugerido es la de definir los lineamientos de diseño, su arquitectura y demás cuestiones técnicas del proyecto de software. Es el que crea documentos de modelos y componentes y especificaciones de interfaces; y valida la arquitectura contra requerimientos documentados por el Analista Funcional. Trabaja en conjunto con éste con el fin de diseñar la arquitectura general de componentes del producto software.

Tester Funcional:
Los Testers Funcionales se encargan de realizar la prueba funcional del producto software, basada en la ejecución, revisión y retroalimentación de las funcionalidades previamente diseñadas para el software. Las pruebas funcionales se hacen mediante el diseño de modelos de prueba que buscan evaluar cada una de las opciones con las que cuenta el paquete informático.

Scrum Master:
El Scrum Master tiene capacidad para gestionar el Proyecto pero su función principal en este modelo es el de motivar al equipo involucrado en el Proyecto. Crea un clima en el cual el equipo trabaje a gusto y sin interrupciones: protege a su equipo. Sin embargo, le deja trabajar libremente. No debe fallar al equipo. No debe faltar a ninguna reunión. Siempre estará ahí, para atender las necesidades del equipo. Es más un “líder” que un “gestor” al uso. Metafóricamente es como el perro pastor de un rebaño de ovejas. Vigila y protege a las ovejas, pero sin molestarlas, sin intervenir prácticamente en sus cometidos.

Project Manager:
Es la persona que tiene la responsabilidad total del planeamiento y la ejecución correcta de cualquier proyecto. El Project Manager debe poseer una combinación de habilidades incluyendo una gran capacidad inquisitiva, de detectar asunciones sin especificar y de resolver conflictos interpersonales. Una de sus tareas más importantes es el reconocimiento de los riesgos que afectan directamente las probabilidades de éxito del proyecto, y la constante medición, formal e informalmente de dicho riesgo a lo largo del ciclo de vida del proyecto. El riesgo se presenta mayormente como incertidumbre. El Project Manager acertado es aquel que enfoca esto como preocupación principal. La mayor parte de los problemas que afectan un proyecto se relacionan de un modo u otro a un riesgo. Un buen Project Manager puede reducir los riesgos significativamente, a menudo adhiriéndose a una política de comunicación abierta, asegurándose de que cada participante significativo tenga

una oportunidad de expresar sus opiniones y preocupaciones. Es el responsable de tomar las decisiones necesarias de manera tal que el riesgo sea controlado y la incertidumbre reducida al mínimo. Cada decisión tomada por el encargado de proyecto debe involucrar un beneficio directo hacia el proyecto.
En el modelo sugerido en el presente trabajo trabaja de manera directa con los Scrum Master estableciendo una guía de trabajo para que sea transmitida a través de ellos al resto del equipo de trabajo en el ambiente On Site y Off Site. El Project Manager es quién lleva el control de los tiempos de ejecución global del proyecto de desarrollo de software distribuido.

4.2.11.2. Actores involucrados en la localización Off Site

Equipo de Programadores:
Equipo de desarrollo del software. Cada equipo estará compuesto por 7 +- 2 personas (es decir: entre 5 y 9 personas, preferiblemente 7). El Scrum Master jamás intervendrá aquí. El equipo debe estar cohesionado y debe promoverse la filosofía de "todos ayudan a todos". El equipo se autogestiona a sí mismo pero siguen una guía establecida por el Scrum Master que está alineado al objetivo principal del software que se está desarrollando y que está supervisado por el Project Manager. En el modelo sugerido se aplica la integración con XP (Extreme Programming) permitiendo la colaboración entre los equipos de programadores en la resolución de problemas de codificación. También son los encargados de ejecutar las integraciones continuas distribuidas del software que se encuentran en los repositorios compartidos y que son utilizados por estos equipos.

Scrum Master:
Idem Scrum Master en localización On Site.

Tester Técnico:
Los Testers Técnicos se encargan de realizar los procesos que permiten verificar y revelar la calidad de un producto software. Las pruebas técnicas son utilizadas para identificar posibles fallos de implementación, calidad, o usabilidad del software que se está desarrollando. Las pruebas técnicas, básicamente, es una fase en el desarrollo de software consistente en probar las aplicaciones construidas. Las pruebas de software se integran dentro de las diferentes fases del ciclo del software dentro de la Ingeniería de software. Así se ejecuta un programa y mediante técnicas experimentales se trata de descubrir que errores tiene. Para determinar el nivel de calidad se deben efectuar unas medidas o pruebas que permitan comprobar el grado de cumplimiento respecto de las especificaciones iniciales del sistema.

Embajadores:
Las comunicaciones y las relaciones de confianza son muy importantes en los procesos de desarrollo de software donde los actores se encuentras distribuidos geográficamente. Los embajadores son aquellas personas, técnicas, funcionales o inclusive de negocios, que se

encargan de afianzar las relaciones entre los ambientes On Site y Off Site. Una de las funciones es la de transmitir con mayor precisión a los equipos de programadores requerimientos del Cliente. Por otro lado un embajador técnico permite colaborar o aclarar cuestiones relacionadas con aspectos netamente técnicos. También es muy importante la figura del embajador de negocios pues se encarga de comunicar al grupo de programadores aspectos relacionados con el contexto del negocio. Los embajadores trabajan junto al Project Manager pues es quién tiene la visión global del proyecto y las necesidades del mismo.

4.2.12. Esquema general de la metodología

En la Tabla 7, se especifica un ciclo de vida de desarrollo de software tradicional donde en cada etapa se detalla los artefactos sugeridos que podrán utilizarse a fin de gestionar y administrar de una manera precisa la evolución del proyecto de desarrollo de software conformado por equipos de desarrollo dispersos geográficamente.

Dicho esquema esta formado por las etapas del ciclo de vida de desarrollo de software sugerido y por aquellos artefactos que mejor se adaptan al proceso de desarrollo de software en ambientes distribuidos desde la visión de la gestión de proyectos ágil y la gestión tradicional.

Ágil, por estar orientado a las personas auto-organizados y la colaboración entre los equipos de desarrollos. Al estar los grupos dispersos geográficamente, es probable que surjan algunos desvíos en el proyecto como consecuencia de algunas interpretaciones o por una débil comunicación entre las partes. Es por esta razón que es importante contar con un modelo de gestión ágil en donde a posibles retrasos que pudieran tener los equipos de desarrollo, el alcance se readapte de una manera rápida y sin mayores inconvenientes. Este es un riesgo que se corre a la hora de desarrollar software con equipos de trabajos dispersos geográficamente y es por este motivo que se cuenta con un modelo reactivo que pueda encausar el desarrollo de una manera efectiva.

Las prácticas y el entorno de trabajo ágiles facilitan la colaboración del equipo, que es necesaria y debe basarse en la colaboración abierta entre todos según los conocimientos y capacidades de cada persona, y no según su rol o puesto.

Las características de los métodos ágiles son que:

- Optimizan la productividad del equipo de desarrollo al contar con un equipo de desarrollo dinámico.
- Se minimizan los riesgos debido a que pueden ser detectados tiempo antes de que se presenten.

Durante el desarrollo de un proyecto surgen circunstancias impredecibles en todas las áreas y niveles. La gestión predictiva confía la responsabilidad de su resolución al Project

Manager. En la gestión ágil, como Scrum, los equipos son auto-organizados, con margen de decisión suficiente para tomar las decisiones que consideren oportunas.

En la propuesta que se plantea en el presente trabajo, la figura del Project Manager se destaca para controlar el proceso de desarrollo de software con el fin de alcanzar el objetivo final establecido en la etapa de Modelamiento de Negocio. Interviene en el resto de las etapas supervisando el avance del proyecto, colaborando con el Scrum Master en la toma de decisiones, pues es el que lleva de manera rigurosa la gestión integral del proyecto.

Mediante la visión tradicional permite determinar los procesos necesarios para organizar las tareas y el personal involucrado en el proyecto de desarrollo de software. Documentar y dejarlo en repositorios. Utilizar algunos artefactos para modelizar el estado de situación como ser Casos de Uso y para establecer como será la arquitectura de software de la aplicación, como Diagramas de Clases y Componentes y la estructura de la base de datos mediante un DER.

Etapa	Ágil	Tradicional	Responsable/s
Modelamiento del Negocio		Visión global Casos de Uso del Negocio	Stakeholders PM Scrum Master Product Owner (Cliente)
Requerimientos	Product Backlog Sprint Backlog Pruebas de aceptación	Casos de Uso Elaboración de Prototipos	PM Scrum Master Analistas Funcional Product Owner (Cliente)
Diseño y Arquitectura		Modelo E-R Diagrama de Clases Diagrama de Componentes Patrones de Diseño	PM Scrum Master Analistas Funcional Arquitectos/Diseñadores
Construcción	XP distribuido Integración continua distribuida		PM Scrum Master Arquitectos Programadores/integradores
Pruebas	Pruebas unitarias Casos de Pruebas Pruebas de integración dist.		PM Scrum Master Analista Funcional Tester
Implementación	Workshop por Sprint terminado (Demo de avance)		PM Scrum Master Product Owner (Cliente) Stakeholders

Iteractivo /Incremental

Sprint (cada 15/30 días)

Tabla 7. Esquema general de la metodología propuesta

En el esquema propuesta existen dos tipos de grupos de procesos. El grupo correspondiente al proceso “estático” y el grupo correspondiente al proceso “dinámico o incremental”.

Dentro del grupo del proceso estático se encuentra la primera etapa, Modelamiento del Negocio y corresponde a aquellas tareas que se realizan al comienzo del proyecto de desarrollo de software. Se determinan la visión global y el alcance del mismo. Es una visión estratégica en la que deben estar involucrados los stakeholders, Project Manager, Scrum Master y el Cliente. Los artefactos utilizados en este grupo son dos: Visión global y Casos de Uso de Negocio. En el primero se determinan los parámetros iniciales del proyecto, como ser cálculo de esfuerzo, equipos de trabajos, distribución de los equipos, riesgos, costos, etc. Mientras que en los Casos de Uso del Negocio se logra "conocer" la organización: misión, funciones, estructura, expertos, tecnología, debilidades, fortalezas; comprender el entorno en el que va a funcionar el sistema, identificar sus procesos, la información, los actores participantes en dichos procesos y los papeles que representan cada uno de ellos, con respecto a cada porción de la información.

Con respecto al grupo del proceso dinámico o incremental, se encuentran las siguientes etapas:

- Requerimientos
- Diseño y Arquitectura
- Construcción
- Pruebas
- Implementación

En la etapa de Requerimientos, es responsabilidad del Cliente describir funcionalmente lo que pretende del software que se desarrollará. Los requisitos del sistema se especifican en casos de usos y en el prototipado de algunas pantallas a fin de establecer si la interpretación del Analista Funcional es la correcta en cuanto a los requerimientos suministrado por el Cliente. Estos documentos formales sirven para tener una versión muy detallada de las necesidades del Cliente. Además de esta documentación, que sirve como documento formal de alto nivel, se confeccionan el product backlog que toma la forma de lista de historias de usuario. Los requisitos del sistema formales se especifican por completo al inicio del proyecto, en cambio el product backlog es un documento vivo, que evoluciona durante todo el desarrollo de forma concurrente con el resto de actividades. Los requisitos del sistema los desarrolla una persona o equipo especializado en ingeniería de requisitos a través del proceso de obtención de información con el cliente, generalmente el Analista Funcional. Esta etapa es la más importante y se utilizan aquellos artefactos que son determinantes para entender y diseñar un modelo de requerimiento que esté de acuerdo a lo que el Cliente solicita para el producto de software. Además es importante utilizar la combinación de artefactos tradicionales y ágiles para que los equipos de trabajo tengan los elementos acordes en caso de producirse un desvío en la ejecución del Proyecto, ya sea por interpretación errónea, por falta de información en los requerimientos o por otros motivos asociados a esta etapa.

En la etapa de Diseño y Arquitectura se diseñan todos los componentes que serán utilizados en el desarrollo del software. Se establece si se reutilizan componentes desarrollados con anterioridad y/o se confeccionan nuevos componentes que serán utilizados por los equipos de programadores remotos. Es importante que cada componente que se utilice esté bien documentado a fin de facilitar la implementación a los Equipos de Programadores. El Arquitecto es el que interpreta técnicamente los aspectos funcionales aportados por el Analista Funcional, es decir, éste le entrega todos aquellos elementos que fueron corroborados junto con el Cliente (pruebas de aceptación, prototipos, etc). En esta etapa se define el modelo de Entidad Relación de la estructura de la base de datos. Por otro lado se especifica el diagrama de clases en el que se describe la estructura del sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño del desarrollo de software, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargaran del funcionamiento y la relación entre uno y otro. También en esta etapa se especifica el diagrama de componentes. Un diagrama de componentes representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre estos componentes. Los componentes físicos incluyen archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables, o paquetes. Debido a que estos son más parecidos a los diagramas de casos de usos estos son utilizados para modelar la vista estática y dinámica de un sistema. Muestra la organización y las dependencias entre un conjunto de componentes. Cada diagrama describe un apartado del sistema. Por último, de ser necesario, se especifican los patrones de diseños que se utilizarán en la construcción del producto software a fin de reutilizar el diseño ya usado y probado satisfactoriamente en implementaciones anteriores.

En la etapa de Construcción del software se programa mediante la codificación del lenguaje de programación establecido para desarrollar el software. Entre los equipos de programadores se distribuyen las tareas de programación de acuerdo a las características técnicas que cada equipo posee. El encargado de distribuir estas tareas es el Arquitecto quién conoce las cualidades técnicas de cada célula de trabajo. Estos equipos de programadores se encuentran ubicados geográficamente fuera del alcance de los principales responsables del proyecto, como es el Cliente, los stakeholders, entre los principales. De todas maneras los programadores trabajan alineados a la estrategia impartida por el Project Manager quien, a su vez, encarga a los Scrum Master el control y motivación de cada grupo de programadores a fin de que se comprometan a cumplir con los objetivos particulares y con los objetivos generales de todo el proyecto de desarrollo de software. Como la comunicación es importante entre los diferentes equipos de programadores se implementa como método de trabajo la Programación Extrema Distribuida (XP Distribuido). La Programación Extrema Distribuida aplica principios de la metodología XP de forma distribuida ya que los miembros del equipo de programadores pueden estar físicamente

alejados. Lo que permite XP a los equipos de programadores es la flexibilidad y la cooperación entre estos equipos. Al estar distribuidos, y si bien se aplican los conceptos que propone XP, es necesario contar con las herramientas para poder lograr una comunicación efectiva entre las partes técnicas involucradas en el proceso de desarrollo de software distribuido. Las tareas de cada equipo de programadores son coordinadas por el Scrum Master de acuerdo a los lineamientos generales impartidos por el Project Manager. Los equipos de programadores realizan la Integración Continua. Esta es una práctica de XP que pone énfasis en el hecho de tener un proceso de construcción y testeo completamente automático, que permita al equipo modificar, compilar y testear un proyecto varias veces en un mismo día. Todas los integrantes del equipo deben integrar su código con la última versión estable por lo menos una vez al día. Martin Fowler [Fowler, 2002] afirma que el desarrollo de un proceso disciplinado y automatizado es esencial para un proyecto controlado, el equipo de desarrollo está más preparado para modificar el código cuando sea necesario, debido a la confianza en la identificación y corrección de los errores de integración. Con el proceso de Integración Continua la mayor parte de los errores de un sistema se manifiestan en el mismo día en el que se integran los módulos intervinientes, reduciendo drásticamente los tiempos para determinar el problema y resolverlo, cada vez que se realiza la integración, los que la hacen deben asegurarse que todas las pruebas se ejecutan correctamente, para que el nuevo código sea incorporado definitivamente. Hacer construcciones regularmente permite tener la posibilidad de obtener feedback continuo y actualizaciones del desarrollo de software distribuido geográficamente. Tal monitoreo permite controlar y seguir el progreso del proyecto, y anticipar eventuales errores y/o discrepancias; lo que proporciona indicadores de la evolución real del código. Para la Integración Continua Distribuida es importante tener una muy buena conectividad a fin de evitar las largas esperas en los procesos de construcción (build) [Miaton, L., 2008]. Es una buena práctica, en general, que el servidor donde se realizan los build se encuentre cerca del equipo de desarrollo principal, que generalmente es el que se encuentra en el ambiente Off Site.

En la etapa de Pruebas del software, el equipo de Testers realiza pruebas unitarias. Las mismas conforman una forma de probar el correcto funcionamiento de un módulo de código determinado. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. Luego, con las Pruebas de Integración Distribuidas, se podrá asegurar el correcto funcionamiento del sistema o subsistema en cuestión. La función principal es escribir Casos de Prueba para cada función no trivial o método en el módulo de forma que cada caso sea independiente del resto. El Scrum Master junto con los testers coordina y configura los Casos de Pruebas correspondientes. Los Casos de Prueba son un conjunto de condiciones o variables bajo las cuáles el Analista Funcional determinará si el requisito de una aplicación es parcial o completamente satisfactorio.

En la etapa de Implementación del producto software, al final de los 30 días, el equipo hace una demostración de la funcionalidad de la pieza (o piezas) de software que se terminaron durante el sprint. Esta demostración se hace directamente en las computadoras del equipo de desarrollo sin mucha preparación previa y coordinada en conjunto con los Scrum Master del ambiente Off Site y los Scrum Master del ambiente On Site. La preparación necesaria se hizo a lo largo del sprint y lo que se presenta es lo que interesa al Cliente y a los Stakeholders.

La duración de esta demostración debe ser por lo menos de 4 horas y de 8 como máximo. El equipo de desarrollo hace las pruebas de funcionalidad para cada uno de los requerimientos aceptados para el sprint en la planeación que se hizo 30 días antes. Si se descubren nuevos requerimientos durante la revisión del sprint, estos se integran al backlog del producto para ser priorizados por el dueño del producto antes de la reunión de planeación del siguiente sprint.

El Cliente debe aprovechar esta reunión para conocer las opiniones de los Stakeholders, debe integrar sus propios requerimientos y adaptar el proyecto con una visión renovada. De esta manera se está cumpliendo con el fundamento de adaptabilidad. Los resultados son controlados a nivel general por el Project Manager quién determina el avance global de todo el proyecto de acuerdo a los hitos establecidos en el proyecto de desarrollo de software.

4.2.13. Simulación de ambiente

Según Miaton [2008] cuando un desarrollo se realiza de manera distribuida geográficamente, en muchas ocasiones el proyecto debe interactuar con otros sistemas y/o fuentes de datos.

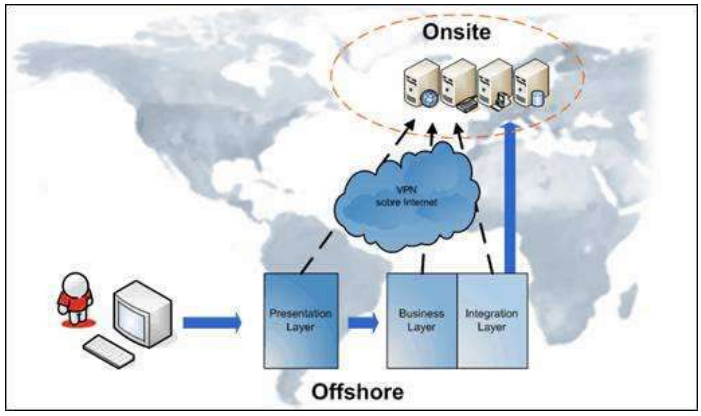

Figura 9. Aplicaciones Off Site que acceden a datos On Site vía VPN[31] [Miaton, L. 2008]

31 VPN: Una red privada virtual o VPN (siglas en inglés de virtual private network), es una tecnología de red que permite una extensión de la red local sobre una red pública o no controlada, como por ejemplo Internet.

Para ello es necesario disponer de una buena conectividad y utilizar canales seguros para mantener la seguridad e integridad de los datos que se transmiten. Esto requiere el acceso por parte del equipo Off Site a los data store On Site mediante una Virtual Private Network (VPN), es decir una línea de comunicación dedicada virtualmente.

Una VPN es una red privada que utiliza un medio de transmisión público y compartido como puede ser por ejemplo Internet. Los mensajes de la VPN transitan sobre la red pública oportunamente cifrados y mediante protocolos seguros como IP Security, PPTP, etc.

Es fundamental tener en cuenta las posibles indisponibilidades de la red y/o de las eventuales carencias de calidad de conexión. Se necesita hacer mitigar estos inconvenientes, de modo que no afecten, o afecten lo menos posible, el trabajo distribuido geográficamente.

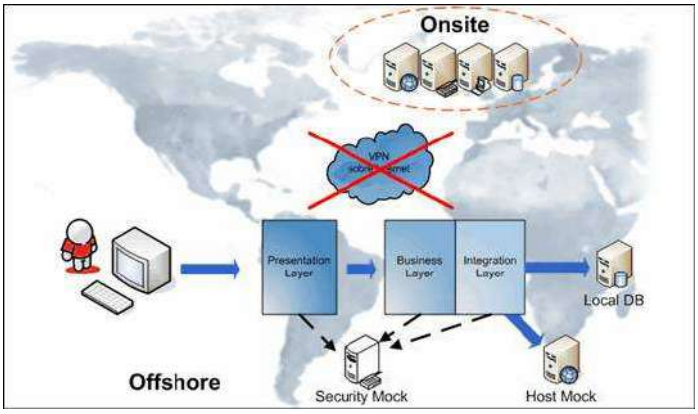

Figura 10. Local DB + Host Mock + Security Mock [Miaton, L. 2008]

Una alternativa es la simulación de los ambientes “Mock Environment”. A continuación se presentan algunas posibles prácticas:

Local DB

Para las aplicaciones en las cuales es necesario recuperar datos de una BD, sería acertado replicar la base, y que el equipo Off Site tenga su propia base donde acceder. Será una tarea del equipo On Site construir las librerías de definiciones y los relativos scripts SQL para su carga. Esto permite al equipo Off Site trabajar en forma independiente del ambiente On Site.

Host Mock

La capa de integración debe "simularse", fingiendo el acceso al host y restituyendo los datos de pruebas oportunamente configurados.

De esta manera es posible desarrollar las partes de la aplicación, sin necesidad de acceder al host. Es importante que tal operación se realice mediante configuraciones y de ningún modo programado.

Security Mock

En ciertos casos es necesario disponer de un “Security Mock”, es decir una "seguridad fingida", que permita la ejecución de la aplicación sin tener que validar la seguridad definida en el ambiente del cliente.

Este caso puede considerarse, como un caso particular del anterior. En cierto modo, cuando se quiere que la aplicación que se esta desarrollando contemple directivas de seguridad (por ejemplo permisos de acceso), y es necesario consultar el esquema de seguridad (cualquiera fuere y se encuentre fuera del alcance del proyecto), entonces se deben poner los controles en forma de configuración (no programático) para su simulación durante el período de desarrollo.

4.3 Herramientas

4.3.1. Introducción

Una vez que sabemos como gestionar un proyecto de desarrollo de software con equipos de trabajo distribuidos geográficamente, una vez que tenemos una metodología que permite organizar dichos equipos de trabajo en conjunto con las tareas asignadas, sólo faltan las herramientas que sirven de apoyo y soporte a toda la metodología y es un factor determinante para llevar adelante la gestión y control de los proyecto de desarrollo de software, como la programación, la integración y las pruebas distribuidas, es decir, el éxito del proyecto depende en gran medida de las herramientas que se utilicen en el proceso de desarrollo del producto software.

De acuerdo a Cerf, S. [2006] existen varias herramientas de comunicación que pueden utilizarse en los proyectos de desarrollo de software con equipos distribuidos geográficamente, pero la más habitual -y también la más eficaz- en la comunicación persona a persona es la mensajería instantánea. Esta herramienta se ha hecho indispensable gracias a sus grandes ventajas. También existen otras opciones, como las conferencias telefónicas (imprescindibles para las comunicaciones de grupo, que son más formales) o las videoconferencias, menos utilizadas En lo que se refiere a la comunicación escrita, el correo electrónico ostenta un indiscutible primer puesto, aunque habrá que seguir de cerca otras propuestas, como las aplicaciones de colaboración tipo wiki, que son muy fáciles de implementar y que permiten crear rápidamente una base de conocimiento estructurada gracias a las aportaciones de todo el mundo.

A menudo también son necesarios los desplazamientos. Tanto al inicio del proyecto como de forma regular, resulta muy útil que los directores del proyecto y los representantes del cliente mantengan alguna reunión con los equipos de desarrollo de software distribuido geográficamente [Cerf, S., 2006].

También hay que tener en cuenta las diferencias culturales, sobre todo cuando se trata de países muy alejados. Una buena comprensión de estos aspectos por parte de los equipos locales aumenta la productividad y los niveles de satisfacción de todas las partes implicadas. Un breve periodo de formación sobre este tema al principio del proyecto resulta una inversión que rápidamente dará sus frutos, aunque no por ello debemos olvidar algo tan crucial como la presencia de un coordinador Off Site.

Desde el punto de vista del proceso de desarrollo de software, existe una relación bien establecida entre el grado de certeza de una tarea y el grado de formalidad de la comunicación en el equipo que resulta adecuado para dicha tarea. En términos sencillos, cuanto menos certera es una tarea, menos formal será el modo de comunicación adecuado.

También se ha establecido que las tareas más complejas requieren más comunicación. La pregunta que es necesario formular es: ¿cuáles son los niveles de certeza y de complejidad de la tarea de desarrollo de software?

En primer lugar, la cuestión de la certeza. Un típico sistema de software complejo incluye una gran cantidad de requerimientos, muchos de los cuales inicialmente se especifican de manera informal. Esto es inherente al proceso de desarrollo de software, debido a que las personas interesadas en el negocio no están capacitadas para brindar especificaciones formales. Se ha comprobado consistentemente la falta de practicidad de especificar formalmente los grandes sistemas. Las especificaciones también cambian con frecuencia durante el curso del proyecto, lo cual agrega un mayor nivel de incertidumbre. La falta de certeza es uno de los motivos para aumentar la adopción de procesos informales "ágiles" en el desarrollo de software.

Pasando a la cuestión de la complejidad, podemos considerar el proceso de desarrollo de software en sí mismo. Los requerimientos informales se deben traducir en especificaciones formales y, en última instancia, en el código fuente y los componentes del software. El proceso de realización de esta traducción es una tarea altamente compleja que demanda una gran cantidad de comunicación entre los arquitectos de TI y las personas interesadas en el negocio. Esto se refleja en los métodos centrados en la comunicación que se aplican a menudo al proceso, como por ejemplo, el diseño participativo [Broady, A., 2010].

Podemos mencionar que los desafíos que enfrentan los equipos de desarrollo de software tienen una naturaleza más sociológica que tecnológica. Esto es consecuencia de la naturaleza incierta y compleja del proceso. La realización de tareas tan complejas e inciertas necesita la colaboración informal. Pero también están los procesos formales y estructurados en todo desarrollo de software que permite el uso de herramientas de colaboración formal.

4.3.2. Herramientas de colaboración formales

Según Broady, A. [2010] la mayor parte de los equipos de desarrollo de software, coestablecidos o no, usan herramientas de colaboración formal. Como ejemplos se pueden mencionar los sistemas de control de versiones, los sistemas de seguimiento de problemas y los sistemas de gestión de requerimientos. Estas herramientas resultan fundamentales para el funcionamiento eficaz del equipo de desarrollo de software. Por lo general, estas herramientas están diseñadas para el desarrollo del software y no se usan en otros procesos del negocio.

Estas herramientas se clasifican como formales porque aplican una estructura sobre la información que almacenan e imponen un proceso para el manejo de la información. Las herramientas formales funcionan sobre información altamente estructurada como por

ejemplo el código fuente, los casos de prueba y los modelos de ingeniería. Tanto los equipos coestablecidos como los formales necesitan herramientas formales.

4.3.2.1. Clasificación de herramientas formales

Rivera, M., [2003], clasifica a las herramientas teniendo como criterio principal la funcionalidad de las mismas:

- *Herramientas de planificación de sistemas de gestión*. Sirven para modelizar los requisitos de información estratégica de una organización. Proporcionan un "metamodelo" del cual se pueden obtener sistemas de información específicos. Su objetivo principal es ayudar a comprender mejor cómo se mueve la información entre las distintas unidades organizativas. Estas herramientas proporcionan una ayuda importante cuando se diseñan nuevas estrategias para los sistemas de información y cuando los métodos y sistemas actuales no satisfacen las necesidades de la organización.

- *Herramientas de análisis y diseño*. Permiten al desarrollador crear un modelo del sistema que se va a construir y también la evaluación de la validez y consistencia de este modelo. Proporcionan un grado de confianza en la representación del análisis y ayudan a eliminar errores con anticipación. Se tienen:
  - Herramientas de análisis y diseño (Modelamiento).
  - Herramientas de creación de prototipos y de simulación.
  - Herramientas para el diseño y desarrollo de interfases.
  - Máquinas de análisis y diseño (Modelamiento).

- *Herramientas de programación*. Se engloban aquí los compiladores, los editores y los depuradores de los lenguajes de programación convencionales. Ejemplos de estas herramientas son:
  - Herramientas de codificación convencionales.
  - Herramientas de codificación de cuarta generación.
  - Herramientas de programación orientadas a los objetos.

- *Herramientas de integración y prueba*. Sirven de ayuda a la adquisición, medición, simulación y prueba de los equipos lógicos desarrollados. Entre las más utilizadas están:
  - Herramientas de análisis estático.
  - Herramientas de codificación de cuarta generación.
  - Herramientas de programación orientadas a los objetos.

- *Herramientas de gestión de prototipos*. Los prototipos son utilizados ampliamente en el desarrollo de aplicaciones, para la evaluación de especificaciones de un

sistema de información, o para un mejor entendimiento de cómo los requisitos de un sistema de información se ajustan a los objetivos perseguidos.

- *Herramientas de mantenimiento*. La categoría de herramientas de mantenimiento se puede subdividir en:
  - Herramientas de ingeniería inversa.
  - Herramientas de reestructuración y análisis de código.
  - Herramientas de reingeniería.

- *Herramientas de gestión de proyectos*. La mayoría de las herramientas de gestión de proyectos, se centran en un elemento específico de la gestión del proyecto, en lugar de proporcionar un soporte global para la actividad de gestión. Utilizando un conjunto seleccionado de las mismas se puede: realizar estimaciones de esfuerzo, coste y duración, hacer un seguimiento continuo del proyecto, estimar la productividad y la calidad, etc. Existen también herramientas que permiten al comprador del desarrollo de un sistema, hacer un seguimiento que va desde los requisitos del pliego de prescripciones técnicas inicial, hasta el trabajo de desarrollo que convierte estos requisitos en un producto final. Se incluyen dentro de las herramientas de control de proyectos las siguientes:
  - Herramientas de planificación de proyectos.
  - Herramientas de seguimiento de requisitos.
  - Herramientas de gestión y medida.

- *Herramientas de soporte*. Se engloban en esta categoría las herramientas que recogen las actividades aplicables en todo el proceso de desarrollo, como las que se relacionan a continuación:
  - Herramientas de documentación.
  - Herramientas para software de sistemas.
  - Herramientas de control de calidad.
  - Herramientas de bases de datos.

4.3.2.2. Integración de herramientas formales

Las herramientas pueden ser integradas de muchas formas, sobre todo cuando se desarrolla software mediante equipos de trabajo distribuidos geográficamente. Para llevar adelante estas integraciones, se crea un número limitado de elementos de configuración de software (documentos, programas o datos) que se manipulan mediante una única herramienta y cuya salida tiene el formato de copia de pantalla y/o documentación gráfica.

El objetivo principal del uso de herramientas, es la integración total de las mismas, lo que permite a los integrantes del proyecto de desarrollo de software compartir toda la

información y el avance del mismo dependiendo de los permisos de acceso que tenga cada actor, independientemente de la ubicación geográfica de cada miembro o célula de trabajo.

El ONGEI[32] [1997] propone cuatro niveles de integración, antes de llegar a la integración total, ellos son:

4.3.2.2.1. Intercambio de datos

La mayoría de las herramientas permiten exportar datos en forma de archivo sin estructura con un formato conocido. Esto permite un intercambio de datos punto a punto entre las distintas herramientas, utilizando normalmente un "filtro" de transmisión intermedio.

En el intercambio de datos punto a punto está en que, a menudo, sólo parte de los datos exportados es utilizable por la herramienta receptora, ya que no fue diseñada para ser totalmente compatible. Además, a medida que evoluciona el software, la necesidad de transferir archivos cada vez que se hace un cambio pequeño puede llevar mucho tiempo. Las versiones pueden quedar "desfasadas" fácilmente, perdiéndose la posibilidad de transferencia, la cual suele ser en un único sentido. No hay posibilidad de que los cambios se reflejen en ambos sentidos y, es difícil hacer comprobaciones cruzadas de documentos y mantener la integridad de la configuración a través de las distintas herramientas que se estén utilizando.

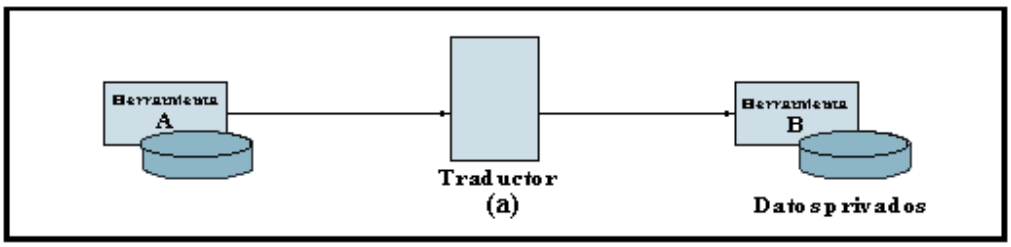

Figura 11. Intercambio de datos [ONGEI, 1997]

4.3.2.2.2. Acceso común a herramientas

Permite al usuario utilizar distintas herramientas de forma similar, por ejemplo a través de un menú desplegable del gestor de ventanas del sistema operativo. En un entorno multitarea, un usuario podría abrir simultáneamente varias herramientas, coordinando manualmente sus entradas y comparando las representaciones de diseño a medida que evolucionan. Por ejemplo, el usuario podría visualizar un diagrama de flujo de datos, un diagrama de estructura, un diccionario de datos y un segmento de código fuente, todos mantenidos por diferentes herramientas. En estos entornos, el intercambio de datos de herramienta a herramienta podría simplificarse llamando al procedimiento de traducción a través de un simple menú o de la selección de una macro.

32 ONGEI: Organismo Nacional de Gobierno Electrónico y Informática del Perú.

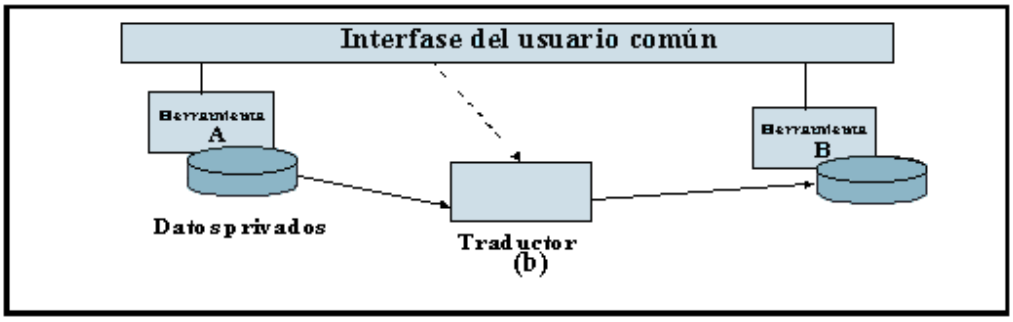

Figura 12. Acceso común a herramientas [ONGEI, 1997]

4.3.2.2.3. Integración de Datos

4.3.2.2.3.1. Gestión común de datos

Los datos de distintas herramientas se pueden mantener en una única base de datos lógica, que puede estar físicamente centralizada o distribuida. Hay una modalidad de fusión que permite combinar el trabajo de varias personas trabajando en diferentes partes de una aplicación.

Aunque los datos generados por las distintas herramientas se gestionan de forma conjunta en el nivel de gestión de datos comunes, las herramientas no conocen de forma explícita las estructuras de datos y la semántica de representación del diseño de las demás. Consecuentemente, se requiere una etapa de traducción (normalmente ejecutada manualmente) para permitir que una herramienta utilice la salida generada por otra.

4.3.2.2.3.2. Datos compartidos

Las herramientas del nivel de datos compartidos tienen estructuras de datos y semántica compatible, pudiendo intercambiar datos sin necesidad de una etapa de traducción. Cada herramienta se diseña para ser compatible con las demás. Por esta razón, la mayor parte del intercambio de datos se da entre herramientas de un único fabricante o en casos en los que se han establecido relaciones estratégicas, entre distintos fabricantes para generar un conjunto de datos integrado, a veces, a petición de clientes importantes.

4.3.2.2.3.3. Interoperabilidad

Las herramientas que combinan las características de acceso común y la capacidad de compartir datos, tienen la capacidad de interoperación. Esto representa el mayor nivel de integración entre herramientas diferentes. Sin embargo, hay otras propiedades del entorno global CASE que se pueden añadir para mejorar la efectividad del proceso de desarrollo de software.

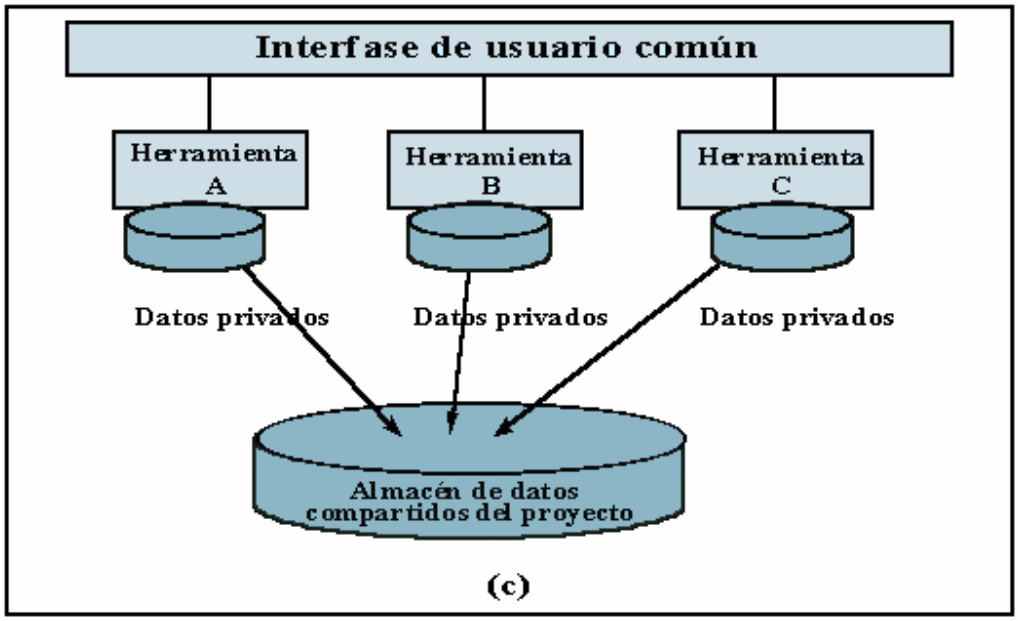

Figura 13. Integración de datos [ONGEI, 1997]

4.3.2.2.3.4. Integración total

Para alcanzar la integración total del entorno utilizando herramientas, se necesitan dos características más: gestión de metadatos y capacidad de control. Los metadatos representan información sobre los datos de ingeniería generados por las distintas herramientas. Esta información incluye:

Definiciones de objetos (tipos, atributos, representaciones y relaciones válidas).

- Relaciones y dependencias entre objetos (p. ej.: un proceso en un diagrama DFD, una entidad única o un fragmento de código de una subrutina).

- Reglas de diseño del software (p. ej.: las distintas formas válidas de dibujar y equilibrar un diagrama de flujo de datos).

- Procedimientos (fases estándar, hitos, informes, etc.) y sucesos (revisiones, finalizaciones, informes de problemas, peticiones de cambios, etc.) del flujo de trabajo (proceso).

Normalmente, la parte de reglas y procedimientos de los metadatos se definen en forma de base de reglas, para facilitar su modificación según evoluciona el proceso de desarrollo del software. Por ejemplo, un nuevo método de diseño podría alterar las reglas de representación y cambiar los estándares del proceso de trabajo seguido hasta el momento.

La capacidad de control permite que cada herramienta pueda notificar al resto del entorno (a otras herramientas, al gestor de metadatos, al gestor de datos, etc.) la ocurrencia de sucesos significativos, así como enviar peticiones para la realización de acciones a otras herramientas y servicios por medio de un activador. Por ejemplo, una herramienta de gestión de configuración que haga una comprobación cruzada de la consistencia de documentos. La capacidad de control ayudará a mantener la integridad del entorno y proporcionará, también, un medio para automatizar procesos y procedimientos estándar. El activador puede estar incorporado en un entorno cerrado o puede estar visible para las distintas herramientas, a través de una interfase de programación y un mecanismo de paso de mensajes.

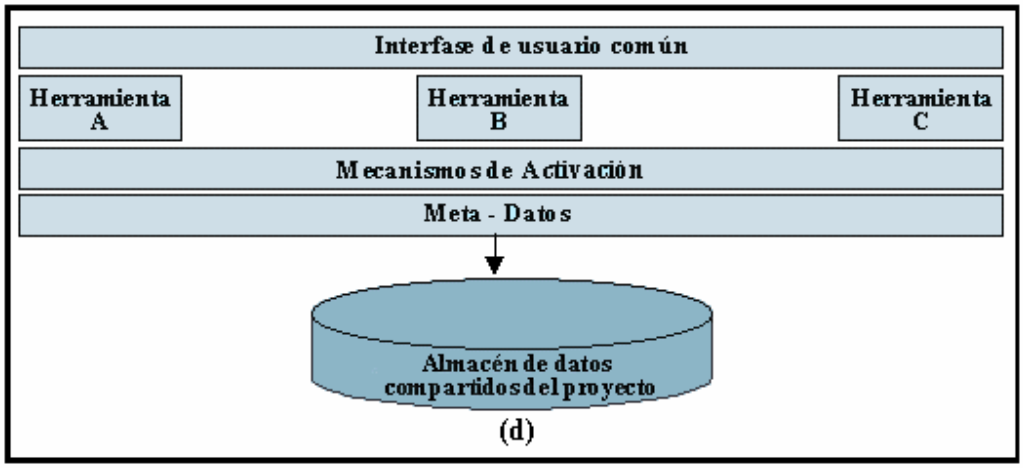

Figura 14. Integración total [ONGEI, 1997]

4.3.3. Herramientas de colaboración informales

Las herramientas de colaboración informal tienen un valor particular para los equipos globales. Estas herramientas permiten que los usuarios trabajen con información menos estructurada y ofrecen amplia libertad respecto del modo en que se utilizan. Las herramientas informales funcionan con información débilmente estructurada, como por ejemplo las conversaciones libres, los bocetos y las opiniones. Las herramientas son genéricas. No son específicas para el desarrollo de software.

Otra clase importante de herramientas es la de las herramientas analíticas. Las herramientas analíticas se usan con frecuencia cuando se debe combinar la información de distintos archivos de información. El ejemplo más obvio es el motor de búsquedas, aunque existen muchos otros, como por ejemplo los tableros de control para la información de gestión. En consecuencia, un criterio importante para el valor de una herramienta de colaboración es su grado de apertura para la integración con las herramientas analíticas.

Según Broady, A. [2010] las tendencias geopolíticas y económicas aumentan el predominio de equipos globales de desarrollo de software. Las tendencias tecnológicas aumentan la

aceptación de herramientas de colaboración informal. El uso de herramientas de colaboración informal es uno de los factores que influyen sobre el éxito de los proyectos globales de desarrollo de software. El valor que aportan las herramientas de colaboración informal a los equipos globales de desarrollo de software se puede ver en tres niveles:

- En un nivel *fundamental*, todos los equipos se comunican equilibrando la comunicación formal e informal. Los equipos globales tienden a depender demasiado de la comunicación formal. La implantación de herramientas que facilitan la comunicación informal puede ayudar a reducir este desequilibrio.

- En un nivel *operacional*, el uso de herramientas de colaboración informal ayuda a aumentar el conocimiento de los demás miembros del equipo, ofrece canales de comunicación más eficaces y compensa la falta de sincronismo.

- En un nivel *estratégico*, las herramientas de colaboración informal ayudan a los gerentes a analizar sus organizaciones. Además, estas herramientas facilitan la retención de conocimientos.

Sin embargo, quizás lo más importante sea que el aumento de la comunicación informal se corresponde con un aumento en la confianza mutua. El uso de este tipo de herramientas fortalece y profundiza las relaciones entre las personas involucradas, lo cual en última instancia lleva a la formación de alianzas estratégicas.

4.3.3.1. Áreas principales a tener en cuenta

Broady, A, [2010] resalta que además de las barreras culturales e idiomáticas, existen otras tres áreas principales que hay que tener en cuenta a la hora de implementar herramientas colaborativas informales en proyectos de desarrollo de software integrado por equipos de trabajo dispersos geográficamente:

- Conocimiento (*awareness*)
- Medio (*medium*)
- Sincronización (*synchronicity*)

Cada una de estas áreas se puede encarar usando diversas herramientas de colaboración informal adecuadas que permitan romper las barreras de comunicación entre los miembros del equipo globales. Las herramientas poseen un valor particular para los equipos globales de desarrollo de software debido a que sirven para cruzar las barreras que enfrentan dichos equipos.

Como muestra la Figura 15, el enfoque básico de usar la telefonía convencional y el correo electrónico es limitado. Dicho enfoque facilita la comunicación a distancia, pero no brinda

una solución al problema del conocimiento, y solo ofrece una solución limitada a las otras áreas. Son necesarias herramientas más avanzadas.

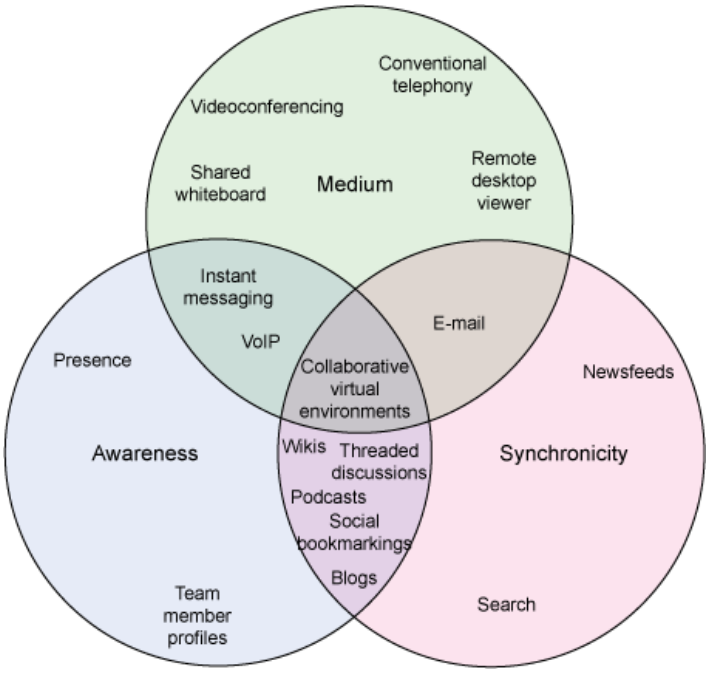

Figura 15. Herramientas de colaboración ayudan a superar las barreras [Broady, A., 2010]

4.3.3.1.1. Conocimiento: Los miembros del equipo deben conocerse entre sí

Ser miembro de un equipo global no significa conocer quiénes son los demás miembros del equipo. A diferencia del equipo coestablecido, en el cual sus colegas son las personas que se sientan a su lado en la oficina.

En un plano más sutil, ser miembro de un equipo global no significa saber algo acerca de la experiencia, las aptitudes o los patrones de trabajo de los demás miembros del equipo.

El problema del conocimiento se puede encarar usando sistemas basados en Internet para ver los perfiles de los miembros del equipo. Al usar estos sistemas, los miembros del equipo se encuentran entre sí en base a las aptitudes, la ubicación geográfica y otros parámetros. Estos sistemas normalmente ofrecen características adicionales, como por ejemplo indicar la hora actual en la zona horaria donde la otra persona trabaja normalmente.

Cuando un miembro del equipo desea contactar a un colega, será útil que sepa si dicho colega se encuentra disponible. A diferencia de los miembros de los equipos coestablecidos, no resulta claro de inmediato si un colega se encuentra en una reunión, está realizando una llamada telefónica o está ocupado en ese momento. Los sistemas de colaboración informal permiten a los miembros de un equipo indicar el estado actual de disponibilidad utilizando el concepto de presencia. Este indicador usualmente está

incorporado a los productos para comunicaciones en tiempo real. Las versiones actuales de estos productos ofrecen las funciones de presencia IM[33], VoIP, videoconferencias, archivos compartidos, y visualización de mesa de ayuda remota integradas, entre las principales.

Existen otras tecnologías que se pueden explorar en esta área. Se ha sugerido que uno de los problemas que enfrentan los equipos globales es que los encuentros oportunistas entre los miembros del equipo son reducidos. Por otro lado estos útiles encuentros a menudo ocurren en zonas neutrales, como por ejemplo pasillos o ascensores, cuando los miembros del equipo trabajan en el mismo lugar. Para los equipos globales, se podrían crear estas zonas neutrales al instalar sistemas de videoconferencia permanente cerca de los refrigeradores o las máquinas de café de los lugares de trabajo de varios miembros del equipo.

4.3.3.1.2. Medio: Los equipos necesitan un sustituto para la comunicación cara a cara

Se considera que la comunicación cara a cara es la forma de comunicación humana más efectiva, con mayor valor psicológico. Es también la mejor manera de construir relaciones de confianza. La mayoría de las personas pueden confirmar esta aseveración a partir de sus experiencias diarias. Este es el motivo por el cual los equipos globales ocasionalmente deberían participar en comunicaciones reales cara a cara, en especial durante las primeras etapas de un proyecto. El uso de tecnología de comunicaciones en tiempo real con multimedios, como por ejemplo la videoconferencia, constituye una alternativa posible.

La participación en una sesión de videoconferencia ocupa el segundo lugar en cuanto a efectividad después de la comunicación cara a cara. Ayuda a construir relaciones de confianza y comprensión mutuas. Esto, a su vez, hace que los miembros del equipo estén dispuestos a explorar ideas innovadoras relacionadas con combinar la información de maneras no obvias. Estos procesos creativos entre los miembros del equipo resultan particularmente útiles cuando se desarrollan nuevos conceptos técnicos o en la etapa de análisis de requerimientos con las personas interesadas del negocio. Probablemente estos procesos resulten particularmente eficaces sólo cuando los participantes se pueden ver entre sí debido a la importancia de la comunicación no verbal. Los miembros de los equipos estarán más dispuestos a discutir soluciones creativas si pueden evaluar mejor las personalidades de sus colegas.

4.3.3.1.3. Medio: Los desarrolladores de software deben poder colaborar visualmente

La ingeniería requiere crear modelos del producto en cuestión. La mejor manera de representar los modelos es en forma visual, mediante diagramas. Esto resulta obvio para aquellas personas que trabajan en las áreas de ingeniería mecánica o civil, debido a la naturaleza tangible del producto, pero no lo es tanto en el contexto de la ingeniería de sistemas. Sin embargo, es igualmente cierto.

33 IM: Instant Messenger (en inglés). Mensajería Instantánea.

Esto ocurre en cualquier reunión técnica de un proyecto de desarrollo de software, seguramente habrá una pizarra cubierta de rectángulos, figures pegadas, nubes y flechas. Es importante mencionar que los diagramas formales desempeñan un rol central. Esto ha sucedido desde el Jackson Structured Programming[34] hasta el enfoque contemporáneo basado en Unified Modeling Language (Lenguaje Unificado de Modelado) o UML. UML es principalmente un lenguaje de diagramas y no de palabras.

Un proyecto de desarrollo de software puede tener éxito solamente si cuenta con un medio de comunicación eficaz, a través del habla y del diseño de diagramas simultáneamente. Durante más de una década, esto se ha podido hacer de manera remota gracias al uso compartido de la pizarra virtual, que brinda un modo electrónico para compartir texto y dibujos igual a las pizarras reales que se usan en las salas de reuniones. Existen dos formas de pizarras electrónicas: una pantalla amplia (por lo general un proyector de video) con "lapiceras" especiales, o el monitor de una computadora convencional que se usa con un "mouse" o una tableta gráfica. La característica del uso compartido de la pizarra virtual está integrada a numerosos productos de comunicaciones en tiempo real.

Además de la necesidad de trabajar sincronizadamente y en colaboración cuando se usan diagramas, los desarrolladores de software y las partes interesadas en el proyecto a menudo deberán tener conversaciones sobre el tema de las GUI[35]. Esto será difícil si los participantes en la conversación no están mirando la misma pantalla en simultáneo. El problema se puede encarar con el uso de herramientas para el uso compartido de escritorio remoto (donde la parte remota también pueda controlar el "mouse" y el teclado). Se trata de una tecnología madura que está integrada a los principales productos de comunicaciones.

4.3.3.1.4. Sincronización: El conocimiento de los miembros del equipo debe estar disponible

Los equipos globales generalmente tienen problemas de sincronización. Con las diferencias de zona horaria y patrones de trabajo que existen, los miembros de un equipo no siempre se pueden comunicar directamente. Una solución posible consiste en alentarlos a registrar sus conocimientos usando herramientas de colaboración informal. Los sistemas actuales en esta área tienen pocas restricciones de ingreso, de manera que los miembros del equipo podrán registrar sus conocimientos con facilidad. A modo de ejemplos importantes, se pueden mencionar los wikis y las herramientas de colocación de marcadores sociales. También se pueden usar los podcasts[36] y los blogs[37]. El tratamiento online de problemas técnicos,

34 También conocido como JSP. Es un método de programación estructurada. Desarrollado en la década de '70 por Michael A. Jackson
35 GUI: Graphic User Interface (en inglés). Pantallas Gráficas del Usuario. Se hace referencia a las pantallas a nivel prototipo que tendrá el producto software.
36 Podcast: el podcasting consiste en la distribución de archivos multimedia (normalmente audio o vídeo) mediante un sistema de sindicación.
37 Blogs: es un sitio web periódicamente actualizado que recopila cronológicamente textos o artículos de uno o varios autores.

usando un sistema de discusiones en cadena (tablero de anuncios o foro electrónico) hace que el resultado sea visible para todos y que todos puedan encontrarlo.

El correo electrónico es una útil herramienta de colaboración informal que también se ocupa del problema de la sincronización. Permite a los participantes entablar una especie de conversación sin tener que estar presentes al mismo tiempo. Debido a su naturaleza privada, resulta más adecuado para cierto tipo de información. Por supuesto, el correo electrónico ya cuenta con un uso generalizado.

Además de la barrera de la sincronización, la organización laxa de muchos equipos globales puede ser problemática. Los recién llegados al proyecto no pueden ser asignados fácilmente a mentores que residen en otros continentes. Ellos podrán accede al historial del proyecto mediante herramientas de colaboración para el uso compartido de conocimientos.

Un enfoque más experimental para el problema de la sincronización sería el uso de un entorno virtual de colaboración. En este tipo de solución, el equipo tiene acceso a un modelo de oficina virtual en 3D. Se accede a los documentos desde ficheros virtuales, y se pueden anexar documentos virtuales de diseño a paredes virtuales.

4.3.3.2. Beneficios del uso de herramientas de colaboración informal

Además de beneficios operativos directos, las herramientas de colaboración informal pueden brindar valor estratégico a largo plazo. En esta área, el equipo global tiene la posibilidad de superar el desempeño de un equipo coestablecido. Debido a que el equipo global debe intercambiar información mediante herramientas, cuenta con más formas de aprovechar al máximo y reutilizar la información. Esto ofrece beneficios que no siempre estarán disponibles para un equipo coestablecido que colabora sin usar herramientas electrónicas.

4.3.3.2.1. Las herramientas de colaboración ayudan a aprovechar los recursos de conocimientos

Se dispone de una amplia variedad de herramientas de colaboración informal para facilitar el almacenamiento y la recuperación de recursos de conocimientos. Entre los ejemplos más conocidos figuran los wikis y blogs. La información que contienen estas herramientas es útil para los proyectos en curso. Además, puede ser retenida, combinada y reutilizada en otros proyectos. Forma una especie de memoria sobre la manera en que se llevaron a cabo los proyectos anteriores. Esta información aumenta la documentación formal de los proyectos. Quizás resulte más exacta y accesible que la documentación formal.

Broady [2010] resume que en el contexto de los proyectos globales de desarrollo de software, esta capacidad de acceder a conocimiento sobre otros proyectos tiene particular importancia. Los equipos offshore a menudo son equipos tercerizados. Una empresa puede

trabajar con múltiples proveedores tercerizados y por lo tanto debe brindarles acceso al conocimiento de sus sistemas. La naturaleza de acoplamiento libre de la organización implica que probablemente los miembros del equipo no tengan conocimiento sobre proyectos anteriores.

El conocimiento almacenado en un sistema de información solo resulta útil si se lo puede encontrar. Esto parece ir en contra del concepto de uso de grandes archivos de información poco estructurados. Sin embargo, la tecnología de búsqueda moderna (como por ejemplo los motores de búsqueda en internet) es claramente capaz de indexar eficazmente grandes colecciones de información poco estructurada. Este tipo de tecnología de búsqueda está comenzando a utilizarse en las empresas, ya que se ha comprobado su aplicación en la información poco estructurada.

La naturaleza basada en internet de muchas herramientas de colaboración informal genera una especie de ecosistema que resulta más valioso que los componentes aislados. Los sistemas basados en internet pueden por sí mismos hacer referencia a otros sistemas similares a través de hipervínculos. Esto es particularmente cierto en el caso de los marcadores sociales. A su vez, los marcadores sociales y otros hipervínculos hacen que los motores de búsqueda sean más eficaces, ya que los asisten en el proceso de calificar los contenidos según las referencias a los mismos, y no en base al contenido en sí.

4.3.3.2.2. La colaboración informal sostiene las alianzas estratégicas

La interacción informal contrarresta la tendencia que tiene la comunicación remota a estar demasiado orientada a las tareas y a ser formal. A través de la colaboración informal, los miembros de un equipo pueden conocerse entre sí y acumular lo que se conoce como capital social. Es este capital social el que permite el florecimiento de una asociación.

Según el Modelo de Intercambio de Recursos entre Socios propuesto por Parise y Henderson [2007], es posible categorizar las asociaciones en función de la complejidad de los aportes realizados por cada uno de los socios. Las asociaciones con mayor valor se producen cuando el compromiso de ambos socios involucra un alto grado de complejidad. En este caso, se puede formar una alianza estratégica. El desarrollo de software es una actividad a través de la cual ambos socios posiblemente realicen aportes altamente complejos a la asociación.

El offshoring normalmente implica una alianza entre un proveedor offshore de servicios de TI y un cliente cuyo negocio central se ubica en otro sector, como por ejemplo servicios financieros. Para que pueda florecer una alianza de estas características, las personas involucradas deben desarrollar relaciones de confianza y capital social, lo cual se puede promover gracias al uso de las herramientas de colaboración informal.

4.3.3.2.3. Las herramientas de colaboración informal facilitan las métricas y el análisis

Con las herramientas de colaboración en funcionamiento, se pueden usar herramientas analíticas para medir la colaboración, brindando así útil información de gestión. Esto brinda a los ejecutivos un valioso “insight” sobre la estructura informal de una organización, algo que es difícil de lograr por otros medios [Broady, A., 2010].

Un enfoque particularmente interesante es el del análisis de redes organizacionales (a veces conocido como análisis de redes sociales). Esta forma de análisis genera una representación virtual de clusters de individuos, con el objeto de mostrar las rutas de las comunicaciones dentro de la organización.

Estos clusters ofrecen un medio para entender de mejor manera cómo es en realidad el trabajo en conjunto del equipo. Estos clusters pueden sentar las bases de las "comunidades de práctica" o facilitar la identificación de los empleados que son fundamentales para el intercambio de información entre los grupos, así como la identificación de expertos en ciertos temas.

El análisis de redes organizacionales (ONA) es posible solamente si se puede registrar la comunicación y la colaboración entre los miembros del equipo. Si esta comunicación se da por medios electrónicos, en especial por correo electrónico o IM, el registro y posterior análisis será bastante sencillo. Si se usa VoIP en lugar de la telefonía convencional, las comunicaciones de voz también estarán abiertas a este tipo de análisis.

Además de las herramientas de comunicación basadas en ONA, las herramientas de uso compartido de conocimiento también se encuentran abiertas al análisis. Como ejemplos simples de esto se pueden mencionar los wikis, los blogs, y las herramientas de marcadores sociales. Todas ellas, en sí mismas, permiten a los gerentes ver cuáles son los miembros del equipo que aportan contenidos y, quizás lo que es más importante, con qué frecuencia los demás miembros del equipo acceden a estos contenidos. Esto ayuda a la gerencia a evaluar ciertos aspectos del aporte realizado por los empleados al proyecto.

Además de evaluar a los individuos, también es posible analizar el proyecto como un todo. Por ejemplo, es posible definir indicadores clave de desempeño, como por ejemplo cuántas vistas de páginas ha habido para el wiki del proyecto en un mes determinado, o cuántas conversaciones en IM se han producido. Si las herramientas se basan en Internet, los datos se podrán recoger mediante herramientas estándar para analíticas web. Estos datos acumulados son probablemente los preferidos por las organizaciones donde existe una preocupación por la privacidad.

El enfoque analítico puede llevarse un paso más adelante al reunir todos los datos de relevancia en un almacén de datos. De este modo, la gerencia podrá ver el grado de colaboración con el proyecto usando un tablero de control.

Según S. Thevenin [2005], los ejecutivos pueden actuar en función de los resultados obtenidos. Los análisis revelarán cuándo se daña la colaboración, en qué casos no se usan las potencialidades de los empleados o cuándo los procesos administrativos evitan que las operaciones sean eficientes. Todo esto facilita la toma de medidas correctivas, que pueden incluir la modificación de roles y responsabilidades para alentar las comunicaciones internacionales.

4.3.4. Herramientas utilizadas en los métodos ágiles y los métodos tradicionales

De acuerdo a Labrin Crawford, B. [2005] en relación a la ingeniería de requerimientos, los métodos ágiles propugnan la activa participación de los usuarios en el mismo lugar de trabajo y por períodos largos de tiempo. Respecto a las prácticas utilizadas, éstas no difieren significativamente de las utilizadas en algunos proyectos tradicionales (Entrevistas, Casos de Uso, Storyboard, Brainstorming, Joint Application Design (JAD), Focus Groups [Escalona, M. J., Koch, 2004]. La diferencia más significativa radica en que los métodos Taylorianos no promueven particularmente las prácticas que aseguren la participación del usuario. En este aspecto cabe recordar que algunos de los principios del movimiento agilista son que “los usuarios y desarrolladores deben trabajar juntos diariamente durante el proyecto” y que “el método más eficiente y eficaz de compartir la información es la conversación cara a cara”. Lo que facilita la definición de las características del sistema a desarrollar, como también la difusión de este conocimiento al equipo de desarrollo dado el estrecho y frecuente contacto con los usuarios. Estableciéndose una diferencia con los proyectos que estructuran los equipos de desarrollo en términos de roles, donde existen analistas, diseñadores y programadores que se dividen las tareas. En este caso, son sólo los analistas quienes se reúnen con los usuarios, generalmente en una sucesión de entrevistas distanciadas temporalmente y donde el objetivo es acordar y documentar las características deseadas.

A través de la colaboración y el diálogo permanente de los participantes, sus modelos mentales y experiencia son compartidos. Es la socialización que se fundamenta en esta posibilidad de poder compartir el tiempo y espacio haciendo colectivo conocimiento tácito.

En los enfoques tradicionales, donde los requerimientos deben ser especificados previos al desarrollo, el equipo a cargo de tareas de diseño y construcción interactúa poco o nada con los usuarios con el objetivo de entender los requerimientos del sistema. Basando su trabajo fundamentalmente en los documentos de especificación.

Este enfoque es adecuado sólo para sistemas cuyos requerimientos se sabe con certeza permanecerán estables. Pero en los escenarios actuales de rápido y constante cambio los requerimientos son inestables La práctica de mantener a los desarrolladores en el mismo lugar de trabajo que los usuarios permite a los métodos ágiles establecer una comunicación permanente para "ajustar" los requerimientos a lo largo del ciclo de desarrollo. Una exigencia para lo anterior es la necesidad de co-localización, lo que para algunas organizaciones no es posible. Existen estudios sobre el uso de métodos ágiles en ambientes de trabajo distribuido que muestran cómo el uso de herramientas para GC[38] ha posibilitado levantar en parte esta restricción.

4.3.5. Herramientas que apoyen la accesibilidad, colaboración y ejecución de los proyectos

Es importante contar con herramientas para el desarrollo de software distribuido, que provean las siguientes capacidades:

- Gestión automática para garantizar las interdependencias entre los componentes y los archivos relacionados, y una veloz actualización de los cambios.

- Realización de pruebas automáticas a los componentes.

- Estandarización de herramientas y procesos para los diferentes sitios. Lo anterior mediante: la utilización de herramientas similares; duplicando al cliente en cada sitio; y estandarizando los procesos de desarrollo.

- Centralizar el acceso a las herramientas a través de: la Web, bases de datos replicadas, ambientes de desarrollo individuales y repositorios centrales.

- Crear y distribuir tutoriales para mejorar la utilización de las herramientas y procesos.

- Según las necesidades del proyecto, desarrollar y/o adaptar las herramientas.

- Infraestructura TI, ésta incrementa la eficiencia y efectividad de la colaboración, y a su vez, aumenta la probabilidad de éxito del proyecto. Los requisitos de la infraestructura TI son:
    - o Acceso rápido a la red de trabajo.
    - o Recursos compartidos como bases de datos, servidores y repositorios del proyecto.
    - o Acceso a las bases de datos y a las herramientas a través de la web.

38 GC: Gestión del Conocimiento

  - Conectividad fácil y rápida entre los sitios remotos, utilizando herramientas colaborativas.

- Tecnologías colaborativas que permitan incrementar la efectividad de la comunicación y la satisfacción personal. Las empresas utilizaron las siguientes herramientas colaborativas:
  - Chat Online: para asuntos urgentes y/o puntuales.
  - Teléfono y teleconferencias: para asuntos urgentes; actualizar la información del estado del proyecto; ayudar a la corrección de errores; y para resolver malos entendidos y conflictos.
  - Aplicaciones compartidas: para ayudar a reparar los errores y proporcionar intercambio de conocimiento.
  - Videoconferencias: para revisar el diseño y efectuar reuniones virtuales.
  - E-mail: para comunicar el estado de las tareas de poca prioridad, aclarar asuntos y enviar código fuente y requisitos.
  - Intranet: para colocar documentos internos.

4.3.6. Herramientas formales para la gestión de proyectos

A continuación se listan un conjunto de herramientas de gran utilidad para el control y gestión de proyectos de desarrollo de software. Todas estas herramientas, por su estructura funcional, permiten gestionar equipos de trabajo distribuidos geográficamente pues tienen componentes colaborativos muy desarrollados que admiten compartir información de avance y otras características entre los responsables de administrar la construcción del producto software.

La mayoría de las herramientas listadas permiten llevar un control de los requerimientos desde una visión ágil, es decir, los item de los Product Backlog como los Sprint Backlog de los proyectos de desarrollo de software.

Gantt PV

Gantt PV es un programa gratuito, de apariencia sencilla y sin grandes complicaciones, para planificación de proyectos, descomposición, representación y seguimiento de tareas sobre diagrama de Gantt.
Descargas disponibles para Windows, MacOS y Linux.

GanttProject

Es una aplicación de escritorio con interfaz similar a MS. Project permite programar y organizar las tareas y asignación de personas y recursos sobre una representación Gantt.
Es una herramienta mucho más ligera que MS Project, pero esto en el ámbito y dimensión de muchos proyectos es más una ventaja que un inconveniente.

Dotproject
Esta solución en entorno web ofrece un marco completo para la planificación, gestión y seguimiento de múltiples proyectos para clientes diferentes, quienes pueden disponer también de acceso para monitorizar la evolución del desarrollo.

TeamWork
Es una herramienta de entorno web para registrar y gestionar los tiempos de diferentes equipos de trabajo en sus respectivos proyectos. Gestión completa de informes de tiempos y costos. Combina gestión de documentos, de equipos y de proyectos.

Planner
Aplicación de escritorio para gestión y seguimiento de proyectos, con descomposición en tareas y sub-tareas, dependiencias, identificación de la ruta crítica, diagramas de Gantt.
Inicialmente desarrollada para Linux, dispone de versión (beta) para Windows.

AgileTrack
Herramienta para planificación y seguimiento de proyectos, de interfaz sencillo. Para desarrollo de software en equipos reducidos con metodologías ágiles, especialmente Extreme Programming.

PPTS
Project Planning and Tracking System (PPTS) es una herramienta de gestión ágil de proyectos para equipos que trabajan con Scrum y/o Extreme Programming. Es un sistema web, accesible con un navegador que puede instalarse sobre servidor Linux o Window.

XPWeb
Plataforma web para gestión de proyectos con Extreme Programming

Trac
Plataforma web para comunicación, gestión y seguimiento de proyectos, que integra un wiki, interfaz de subversión para la gestión de versiones, seguimiento de proyecto y sistema de tickets para gestionar y registrar tareas, bugs, etc.

TUTOS
Herramienta web de código abierto y uso gratuito para la gestión de pequeños grupos de trabajo o departamentos. Incluye calendario, gestión de equipos, directorio de personas, gestión de incidencias, registros de tiempo, listas de seguimiento, etc

Solodox
Servicio de software que permite editar y compartir con el equipo y demás interesados planificaciones Gantt.

ToDoList
ToDoList es una herramienta gratuita muy simple y efectiva para la gestión de proyectos en entornos ágiles.

Clocking IT
Es un gestor de proyectos y tareas, con control de tiempos, generador de informes, repositorio de ficheros, agenda, chat, notificaciones y RS.

X-Man
X-Man (Extreme Manager) es una herramienta fácil para gestión y seguimiento de proyectos ágiles.

Mindquarry
Sistema basado en web para la gestión de grupos de trabajo, entornos colaborativos, proyectos.

OpenProj
Es un programa de escritorio para la gestión de proyectos: gratuito, open source, con versiones para Linux, Unix, Mac y Windows; compatible con ficheros MS Project y con todas las funcionalidades que ofrece Project.

Project Dune
Sistema sobre web para integrar todos los procesos y documentación del ciclo de desarrollo.

Activity Manager
Programa Open Source para registrar y clasificar por tareas y sub-tareas los tiempos de trabajo del equipo de un proyecto.

PrjPlanner
Herramienta para la auto-gestión ágil de equipos de programación pequeños. Está inspirada en el concepto de backlog de Scrum.

Project2Manage
Se trata de un servicio web, con funcionalidades simples pero que pueden ser suficientes para el registro y la comunicación de actividades entre los miembros de un equipo de trabajo.

Collabtive
Es una plataforma on-line para gestión de proyectos y colaboración de equipos de trabajo. Es open source, y se puede utilizar gratuitamente con licencia GNU. Requisitos: Linux, Apache y PHP5.

RedMine
Sistema multi-plataforma, programado con Ruby on Rails, open source con licencia GPL, con un interfaz limpio y unas funcionalidades asombrosas para gestión de proyectos.

iceScrum
Con el mismo interfaz para todos los roles, ofrece las opciones de operación, consulta, estimación de historias de usuario activas o no, según el usuario sea propietario del producto, gestor, equipo o interesado.
Incluye listas de historias de usuario (backlog), de asuntos, de problemas y de pruebas; un chat en línea, un juego de cartas con el que el equipo puede hacer estimación de poker de las historias propuestas en el backlog, etc.

Google Sites
En muchos casos puede ser más que suficiente. Es una solución útil y simple, que consiste en componer el punto de información y registro de información, a la medida del proyecto, integrando diferentes Google apps

FVE (extensiones para dotProject)
FVE Project Manager es una adaptación del programa libre para gestión de proyectos: dotProject realizada con licencia GPL v3.

Opengoo
Sistema web para gestión de equipos de trabajo. Desde el área de administración se pueden ajustar los permisos de cada usuario, de forma individual o por grupos, y para cada área de trabajo, incluso para que no resulte visible.

Sprintometer
Es un programa windows, gratuito, contenido en un único fichero ejecutable, que no necesita instalación, para gestión, medición y seguimiento de programas con modelos ágiles tipo Xp o Scrum.

4.3.7. Herramientas formales para las fases de Diseño, Construcción, Pruebas e Implementación

4.3.7.1. Herramientas para modelar diagramas entidad relación

El Modelo Entidad-Relación, también conocido como DER (diagramas entidad-relación) es una herramienta de modelado para bases de datos mediante el cual se pretende 'visualizar' los objetos que pertenecen a la Base de Datos como entidades (se corresponde al concepto de clase, cada tupla representaría un objeto, de la Programación Orientada a Objetos) las cuales tienen unos atributos y se vinculan mediante relaciones. Es una representación conceptual de la información. Mediante una serie de procedimientos se puede pasar del modelo E-R a otros, como por ejemplo el modelo relacional.

DB Designer Fork

Posee una interfaz que permite modelar las relaciones entre tablas permitiendo generar scripts de SQL para bases de datos relacionales como Oracle, SQL Server, MySQL, FireBird, SQLite y PostgreSQL

MySQL Workbench

Esta herramienta permite diseñar visualmente relaciones de entidad relación. El propósito principal de esta herramienta es para que sea utilizada para bases de datos MySQL.

Open System Architect (OSA)

Básicamente esta herramienta permite modelar un sistema. OSA, actualmente, ayuda a modelar datos de una manera física y lógica con el lenguaje UML. Es una herramienta open source que se distribuye bajo licencia GPL.

4.3.7.2. Herramientas para diseñar diagramas de clases, diagramas de componentes y casos de uso

yUML

Es un servicio online para crear diagramas de clase y de casos de uso. Este servicio puede llamarse desde un blog o página web (pasando la descripción textual del modelo a mostrar como parte de la URL) para visualizar automáticamente el modelo indicado.

UML Graph

Dibuja automáticamente diagramas de clase y de secuencia. Para los de clase utiliza la sintaxis Java con anotaciones que después la herramienta convierte a “specifications Graphviz”. Para los diagramas de secuencia se utiliza un enfoque diferente: se usan pic macros para definir el diagrama y después el programa pic2plot convierte las macros en archivos gráficos.

Enterprise Architect
La versión Corporativa de esta herramienta es una excelente solución que permite modelar en lenguaje UML todos los procesos de desarrollo de software.

4.3.7.3. Herramientas para realizar integración continua

La integración continua es una metodología informática que consiste en hacer integraciones automáticas de un proyecto lo más a menudo posible para así poder detectar fallos cuanto antes. Entendemos por integración la compilación y ejecución de tests de todo un proyecto.

Apache Continuum
Apache Continuum es un servidor de integración continua a nivel corporativo. Posee características como construcción automática, administración de versiones, seguridad basada en roles.

Hudson
Hudson es un sistema de integración continua de fácil uso. Esta herramienta permite a los desarrolladores integrar los cambios surgidos en el proyecto y a los usuarios obtener información de las construcciones. Es una herramienta de construcción automática que construye la aplicación de una manera continua incrementando la productividad.

CruiseControl
Es una herramienta comúnmente utilizada en integración continua que cada cierto tiempo, o cuando hay cambios en el gestor de versiones (por ejemplo CVS o Subversion), hace una compilación y ejecuta tests y una vez acaba presenta el resultado.

4.3.7.4. Herramientas para el control de versiones

Una versión, revisión o edición de un producto, es el estado en el que se encuentra dicho producto en un momento dado de su desarrollo o modificación. Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Los sistemas de control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas.

CVS
El Concurrent Versions System (CVS), también conocido como Concurrent Versioning System, es una aplicación informática que implementa un sistema de control de versiones: mantiene el registro de todo el trabajo y los cambios en los ficheros (código fuente principalmente) que forman un proyecto (de programa) y permite que distintos desarrolladores (potencialmente situados a gran distancia) colaboren.

Subversion

Subversion es un software de sistema de control de versiones diseñado específicamente para reemplazar al popular CVS. Una característica importante de Subversion es que, a diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en un instante determinado. Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintas computadoras.

SourceSafe

Microsoft Visual SourceSafe (también conocido por sus siglas VSS) es una herramienta de Control de versiones que forma parte de Microsoft Visual Studio aunque está siendo sustituida por el Visual Studio Team Foundation Server.

Bazaar

Bazaar es un sistema de control de versiones distribuido patrocinado por Canonical Ltd.[39], diseñado para facilitar la contribución en proyectos de software libre y opensource. Bazaar puede ser usado por un usuario único trabajando en múltiples ramas de un contenido local, o por un equipo colaborando a través de la red.

Plastic SCM

Plastic SCM es un Sistema de Configuraciones Software (en inglés Software Configuration Management) desarrollado por la empresa española Códice Software. Como objetivos fundamentales, Plastic trata de dar un mayor soporte al desarrollo paralelo, creación de ramas, fusión (merge) de ramas y seguridad.

4.3.7.5. Herramientas para la ejecución de pruebas unitarias

En programación, una prueba unitaria es una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. Luego, con las Pruebas de Integración, se podrá asegurar el correcto funcionamiento del sistema o subsistema en cuestión. La idea es escribir casos de prueba para cada función no trivial o método en el módulo de forma que cada caso sea independiente del resto.

JUnit

JUnit es un conjunto de clases (framework) que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera.

39 Canonical Ltd. es una empresa privada fundada y financiada por el empresario sudafricano Mark Shuttleworth, para la promoción de proyectos relacionados con software libre.

PHPUnit
Entorno de prueba para realizar pruebas unitarias en PHP.

CPPUnit
CPPUnit es un módulo framework de pruebas unitarias para C++, descrito como una adaptación de JUnit en C++.

NUnit
NUnit es un entorno open source de Pruebas de unidad para Microsoft .NET y Mono. Sirve al mismo propósito que JUnit realiza en el mundo Java, y es uno de muchos en la familia xUnit.

NUnit.Forms
Es una expansión al entorno núcleo NUnit y es también open source. Esto busca concretamente ampliar NUnit para que sea capaz de manejar pruebas de elementos de interfaz de usuario en Windows Forms.

NUnit.ASP
es una expansión al entorno núcleo NUnit y es también open source. Es capaz de manejar pruebas de elementos de interfaz de usuario en ASP.NET.

4.3.8. Propuesta de herramienta colaborativa integrada

Analizando las secciones anteriores donde se detallaron las características de las herramientas formales y de aquellas herramientas que son de soporte para la gestión individual denominadas como herramientas informales, observamos que existen una gran variedad de herramientas que permiten ayudar a los equipos de trabajo distribuidos geográficamente en la gestión y desarrollo de los proyectos de construcción de software.

En cada etapa del ciclo de vida de desarrollo de software, como en la gestión total del proyecto y para cada uno de los artefactos definidos en el presente trabajo se han definido herramientas que sirven de soporte a la metodología propuesta. Sin embargo, la utilización de esas herramientas conlleva a la implementación de una diversidad de ellas ya que por cada etapa y por cada actor participante se deberían instalar aquellas herramientas que mejor se ajusten ya sea por las características del proyecto o por como se han distribuido los equipos de trabajo. Implementar diferentes herramientas ya sea para gestión, desarrollo, pruebas, integración, etc., dificulta por un lado la configuración y mantenimientos y por otro lado el control total del proyecto pues en algún punto se pierde el flujo de trabajo al utilizar estas herramientas heterogéneas.

Una solución a esta problemática es la de implementar una herramienta donde se incluyan todos los procesos que intervienen en la construcción del producto software con equipos de

trabajo distribuidos geográficamente. Para lograr este objetivo es determinante contar con la metodología adecuada, como la que se presenta en el actual trabajo, ya que de esta manera se determinan las fases y los artefactos que intervienen en las mismas junto con los actores que forman parte de todo el proyecto. Una vez identificado todo este conjunto de elementos que hacen a la metodología propuesta, el paso siguiente es la construcción o integración de otras aplicaciones en una herramienta colaborativa única que sirva para centralizar toda la información y permita a cada uno de los actores llevar un control de los procesos que forman parte del proyecto de desarrollo de software distribuido.

Esta nueva herramienta colaborativa estará conformada por diferentes módulos cuya información estará almacenada en un único repositorio de datos pudiendo acceder aquellos actores que tengan los permisos adecuados. La herramienta estará implementada en un solo lugar físico (con su esquema de redundancia correspondiente) y será accedida desde cualquier parte del mundo a través de una VPN corporativa asegurando el canal de comunicación entre todas las partes que conforman el equipo de trabajo.

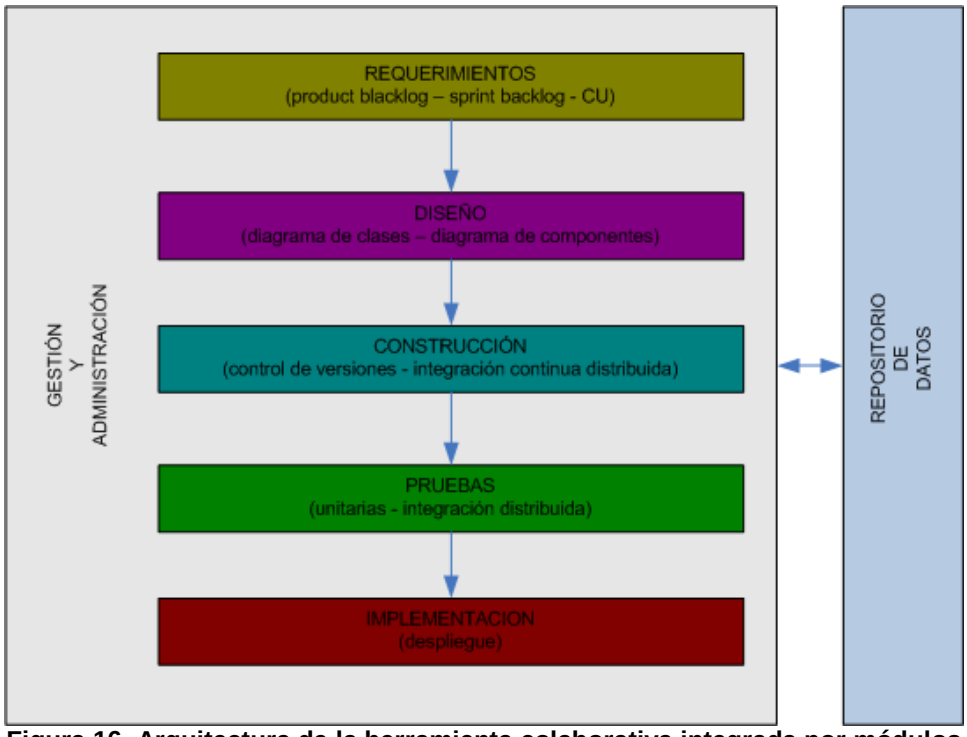

Figura 16. Arquitectura de la herramienta colaborativa integrada por módulos

5. CONCLUSIONES

A partir del análisis efectuado se puede destacar la importancia de desarrollar una metodología de desarrollo de software aplicable a entornos de equipos de trabajo distribuidos geográficamente ya que permite llevar la gestión y los procesos de desarrollo de una manera aplicada y organizada utilizando herramientas como soporte a la administración general del proyecto.

Por lo tanto del trabajo realizado en esta tesis se puede extraer varias conclusiones:

- Se diseñó una metodología flexible de desarrollo de software para equipos de tareas trabajando en ambientes virtuales tomando como base las características de las metodologías ágiles y tradicionales que mejor se adaptan a entornos de trabajo dispersos geográficamente.

- Se investigó la existencia de metodologías de similares características a la formulada en el presente trabajo llegando a la conclusión que no hay una metodología establecida para afrontar desarrollos con equipos de trabajo dispersos geográficamente. Por lo tanto el desarrollo de una metodología estructural sirve para guiar a los equipos de trabajos en los procesos de desarrollo de software distribuidos.

- Al evaluar las fortalezas y debilidades de las metodologías existentes permitió determinar cuales eran las características principales de cada una. Al tener un conocimiento concreto de las metodologías de desarrollo de software, tanto ágiles como tradicionales, aportó para desarrollar la estructura general de la metodología propuesta en el presente trabajo y de esta manera avanzar en el diseño de la misma.

- Al diseñar una metodología basada en 3 capas, en primera instancia se analizaron diferentes mecanismos de control y gestión de proyectos de software que mejor se adaptan a los procesos de desarrollo de software trabajando en ambientes virtuales. En la actualidad existen algunas metodologías de desarrollo, aplicadas al desarrollo del software, que incorporan en mayor o menor medida muchos de los conceptos relacionados con el manejo de las personas. Pero lo cierto es que todas estas metodologías proporcionan un enfoque incompleto del problema ya que lo que proponen muchas veces no es aplicable a proyectos distribuidos geográficamente. Además, los estándares definidos para la administración de proyectos presentan el tema de la administración de las personas pero no brindan detalles acerca de técnicas a utilizar, y por ello muchas veces es muy difícil de implementar un manejo correcto del factor humano. Para ello se evaluó como alternativa de gestión de

proyectos distribuido el uso de Scrum, ya que permite centrarse en las personas más que en los procesos y al ser una metodología ágil está mejor preparada para soportar cambios en los procesos de desarrollo por motivos diferentes ya que las distancias y las interrelaciones entre los miembros del equipo de desarrollo es una de las causas que la metodología propuesta trata de resolver.

- Se definió una matriz comparativa entre las dos orientaciones, la visión ágil y la visión tradicional, que sirvieron como base para establecer y diseñar la metodología de desarrollo de software con equipos de trabajo dispersos geográficamente. En esta matriz se detallan los aspectos principales de las dos orientaciones agrupadas por Aplicación, Gerenciamiento de proyecto, Aspectos técnicos, Personal y Cultura. Con estos grupos de características se estableció una visión clara de las acciones y procedimientos que se llevan a cabo en cada una de las orientaciones. A partir de estos se identificaron sus cualidades principales estableciendo un criterio para adecuarlo a la metodología de desarrollo de software en ambientes virtuales del presente trabajo. Al tener una visión particular de cada orientación se llegó a la conclusión que la combinación de las mejores características de ambas visiones en una única metodología que ayude a los procesos de desarrollo de software era el camino correcto pues ambas visiones ya se encuentran en uso en muchos proyectos de desarrollo de software de diferentes características pero previo a este trabajo pocas veces se ha analizado en detalle como adaptar estas metodologías a un entorno de producción con equipos de trabajo dispersos geográficamente.

- Se definieron un conjunto de indicadores que permiten evaluar la incorporación de una metodología de desarrollo de software con equipos de trabajos dispersos geográficamente. El modelo propuesto de indicadores contempla la identificación de aquellas variables que mejor describen el entorno, es decir, las características técnicas, ambientales y profesionales que rodean al desarrollo de un proyecto de software distribuido geográficamente.

- En los entornos de desarrollo de software con equipos de trabajos dispersos geográficamente, las herramientas colaborativas juegan un rol importante en todo el proceso pues sirven de apoyo a las dos capas superiores, es decir a la gestión y control y al desarrollo de software propiamente dicho. Para ello se analizaron diferentes herramientas desde una visión formal y desde una perspectiva informal. Desde la perspectiva informal se identificaron todas aquellas herramientas que sirven de ayuda a la comunicación e interrelación entre las personas del proyecto como apoyo a la gestión individual, destacándose el uso del chat, de repositorios, de espacios colaborativos, el email y de foros de discusión para llevar a adelante las comunicaciones interpersonales. Mientras que la visión formal de las herramientas está orientada a la gestión integral de todo el proceso de desarrollo de software

llevada a cabo por cada actor ejecutante. Como ejemplos se pueden mencionar los sistemas de control de versiones, los sistemas de seguimiento de problemas y los sistemas de gestión de requerimientos. Estas herramientas resultan fundamentales para el funcionamiento eficaz del equipo de desarrollo de software. Por lo general, estas herramientas están diseñadas para el desarrollo del software y no se usan en otros procesos del negocio. Por lo tanto, estas herramientas se clasifican como formales porque aplican una estructura sobre la información que almacenan e imponen un proceso para el manejo de la información. Las herramientas formales funcionan sobre información altamente estructurada como por ejemplo el código fuente, los casos de prueba y los modelos de ingeniería.

- En el análisis de las herramientas colaborativas existentes en el mercado se identificaron aquellas de acuerdo al grupo de actividades a las que pertenecen. Se determinó un listado de estas herramientas para la gestión de proyectos, para modelar diagramas entidad relación, para diseñar diagramas de clases, diagramas de componentes y casos de uso, para realizar integración continua, para el control de versiones y para la ejecución de pruebas unitarias. Si bien estas herramientas son muy potentes individualmente, en conjunto se pierde el control y la usabilidad de las mismas. La utilización de esas herramientas conlleva a la implementación de una diversidad de ellas ya que por cada etapa y por cada actor participante se deberían instalar aquellas herramientas que mejor se ajusten ya sea por las características del proyecto o por como se han distribuido los equipos de trabajo. Implementar diferentes herramientas ya sea para gestión, desarrollo, pruebas, integración, etc., dificulta por un lado la configuración y mantenimientos y por otro lado el control total del proyecto pues se en algún punto se pierde el flujo de trabajo al utilizar estas herramientas heterogéneas. Por lo tanto para solucionar lo antes mencionado se propuso el desarrollo o la integración de varias de las herramientas en un única pudiéndose tener un solo marco de interfase de usuario y un único repositorio de datos.

6. FUTURAS LÍNEAS DE INVESTIGACIÓN

A partir del presente trabajo de tesis, existen aspectos que quedan fuera del alcance planteado y que forman parte de las líneas de investigación futuras. Entre ellas las más importantes son:

- Incorporar en la metodología propuesta consideraciones sobre las áreas del conocimiento mencionadas en el PMBOK y que no se detallaron en el presente trabajo: Gestión del Tiempo en Proyectos, Gestión de la Calidad en Proyectos, Gestión de Costos en Proyectos, Gestión del Riesgo en Proyectos y Gestión de la Procura (Logística) en Proyectos.

- Identificar, de acuerdo al tipo de proyecto de desarrollo de software distribuido, otros artefactos que mejor se adapten al tipo de entorno diseñado geográficamente para crear un producto software utilizando células de trabajo dispersos geográficamente.

- Profundizar en los aspectos metodológicos discutidos en el presente trabajo a fin de determinar nuevas soluciones dependiendo del modelo de distribución de equipos de trabajo dispersos geográficamente.

- Profundizar tanto en las aptitudes, actitudes y en los aspectos sicológicos de cada uno de los miembros que conformarán los equipos de trabajos a fin de contar con los perfiles adecuados para llevar adelante proyectos de desarrollo de software distribuidos geográficamente.

- Definir que herramientas son las adecuadas para integrarlas en un única herramienta y de esta manera poder ejercer una gestión de proyectos en todas las etapas (requerimientos, diseño, construcción, pruebas e implementación) más eficiente y eficaz.

7. CITAS Y REFERENCIAS BIBLIOGRÁFICAS

- Abdallah García, Angélica B. (2004). Asociación Argentina de Teletrabajo, Buenos Aires, Argentina.
- Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J. (2002), “Agile software development methods Review and analysis”. VTT Publications.
- Aguadero, F. (1997). La sociedad de la Información. Acento Editorial. Madrid.
- Álvarez, José R. y Arias, Manuel (2002). Análisis y definición de requisitos. Diseño, propiedades y mantenimiento del software. Universidad Nacional de Educación a Distancia. España.
- Ambler, S. W. y Jeffries, R. (2002). Agile Modeling for Extreme Prog. W/Ws. Publisher: John Wiley & Sons.
- Bannon y Schmidt (1889). ECSCW ‘89. Proceedings of the First European Conference on Computer Supported Cooperative Work, London, 1989.
- Bauer, F. L (1972). Software Engineering Information Processing. Amsterdam. North Holland.
- Beck, K. Extreme Programming Explained. Embrace Change (1999), Pearson Education. Traducido al español como: “Una explicación de la programación extrema. Aceptar el cambio” (2000), Addison Wesley.
- Bellagio, D. E y Milligan, T. J. (2005). Software Configuration Management Strategies and IBM Rational ClearCase. Ed. Addison Wesley, 2 da. Edición.
- Boehm, Barry (1988). “A Spiral Model of Software Development and Enhancement”, from Proceedings of an International Workshop on the Software Process and Software Environments.
- Brinck, T. (1998). What is the Groupware? Explorador Internet: http://tecfa.unige.ch/staf/staf-e/paraskev/staf14/ex8/article2.html (última consulta 20/09/2009)
- Broady, Alan F. (2010). Herramientas de colaboración informal para los equipos globales de desarrollo de software Informal. http://www.ibm.com/developerworks/ssa/rational/library/09/informalcollaborationtools forglobalsoftwaredevelopmentteams/index.html (última consulta 10/05/2010).
- Brooks, Frederick P. Jr, 1987 "No silver bullet: Essence and accidents of software engineering", Computer, vol. 15, no. 1, pp. 10-18.
- Castells, Manuel (1995), La ciudad informacional: tecnologías de la información, reestructuración económica y el proceso urbano-regional, Alianza Editorial, Madrid, 504 p. Introducción y capítulo 1.

- Castellón, Lucía (2001), Las múltiples dimensiones de la brecha digital, Reflexiones Académicas número 13, Universidad Diego Portales.
- Cerf, Stephane (2006). Los cinco puntos clave en el offshore. http://www.emprendedores.cl/desarrollo/mantenedores/art_indice.asp?art_id=131 (última consulta 18/11/2009)
- Coad P., Lefebvre E., De Luca J. (1999). Java Modeling In Color With UML: Enterprise Components and Process. Prentice Hall.
- Cockburn, A. (2002). Agile Software Development. Addison-Wesley. 278 p.
- Cockburn, A. (2000). Characterizing People as Non-Linear, First-Order Components in Software Development. http://alistair.cockburn.us/Characterizing+people+as+non-linear%2c+first-order+components+in+software+development (última consulta 15/04/2009)
- Comisión Europea (1993 a), Libro blanco Sobre el crecimiento, la competitividad y la ocupación (White Paper on Growth, Competitiveness, and Employment), Comisión Europea, Bruselas.
- Comisión Europea (1994 b), Europa y la sociedad global de la información (Informe Bangemann), Recomendaciones al Consejo Europeo, comisión europea, Bruselas, 35 p.
- Comisión Europea (1996 c), Libro Verde vivir y trabajar en la sociedad de la información: prioridad para las personas, Suplemento 3/96 de la Unión europea, Comisión europea, Luxemburgo, 32 p.
- Comisión Europea (1997 d), Libro Verde sobre la convergencia de los sectores de telecomunicaciones, medios de comunicación y tecnologías de la información y sobre sus consecuencias para la reglamentación en la perspectiva de la sociedad de la información, Comisión Europea, Bruselas, 45 p.
- Diaz M., A. (2000). E-business: Tecnología de información y redes de negocios. Debates IESA. Vol. V. No. 4.
- Espinoza Villareal, M. (2000). Estrategias de moderación como mecanismo de participación y construcción del conocimiento en grupos de discusión electrónicos. ITESM. México.
- Escalona, M. J., Koch, N. (2004). Ingeniería de Requisitos en Aplicaciones para la Web: Un estudio comparativo. http://www.pst.informatik.uni-muenchen.de/personen/kochn/ideas03-escalona-koch.pdf (última consulta 01/07/2010)
- Fogel, K. (1999). Open Source Development with CVS: Learn How to Work With Open Source Software. Ed. Coriolis Group Books; illustrated edition.

- Fowler, M., Beck, K., Brant, J. (1999). Refactoring: Improving the Design of Existing Code. Addison-Wesley.
- Fowler, Martin. (2006). Using an Agile Software Process with Offshore Development. http://martinfowler.com/articles/agileOffshore.html (última consulta 05/07/2009)
- Gabardini, J y Campos, L (2004). Balanceo de Metodologías Orientadas al Plan y Ágiles. Herramientas para la Selección y Adaptación. http:// www.rmya.com.ar/Download/Paper_BMOPA.pdf (última consulta 16/10/2009)
- Grupos ISSI (2003). Ingeniería del Software y Sistemas de Información. Colaboradores. Universidad politécnica de Valencia. Taller realizado en el marco de las VIII jornadas de Ingeniería del Software y Bases de Datos.
- Guardiola, U. (1998). El control de gestión y sus indicadores. Bogotá: Incolda.
- Haraway, D. (1995). Manifiesto para cyborgs: ciencia, tecnología y feminismo socialista a finales del siglo xx, en Ciencia, cyborgs y mujeres. La reinvención de la naturaleza. Cátedra, Madrid, p. 281-283
- Hernandez Arias, Aymará (2002), La tecnología de Trabajo Colaborativo en el contexto universitario. http://www.ucla.edu.ve/dac/investigaci%F3n/compendium6/Tecnologia%20de%20trabajo%20colaborativo.htm (última consulta 21/09/2009).
- Highsmith J., Orr K. (2000). Adaptive Software Development: A Collaborative Approach to Managing Complex Systems. Dorset House.
- IBM (2010). Herramientas de colaboración informal para los equipos globles de desarrollo de software informal. http://www.ibm.com/developerworks/ssa/rational/library/09/informalcollaborationtools forglobalsoftwaredevelopmentteams/index.html (última consulta 20/04/2010).
- Kroll, P y Maclsaac, B (2006). Agility and Discipline Made Easy: Practices from OpenUP and RUP. Ed. Addison-Wesley Professional, 1ra edición.
- Jeffries, R., Anderson, A., Hendrickson, (2001). C. Extreme Programming Installed. Addison-Wesley.
- Labrin Crawford, B. (2005). Métodos Ágiles: Enfatizando el Tratamiento del Conocimiento Tácito sobre el Explícito. Escuela de Ingeniería Informática, Facultad de Ingeniería Pontificia Universidad Católica de Valparaíso, Chile.
- Larman, C. (2003). Agile and Iterative Development: A Manager's Guide. Addison-Wesley Professional, 1st edition.
- Laudon y Laudon. (2000). Administración de los Sistemas de Información. Organización y Tecnología. 3ra edición. Prentice Hall. México.

- Letelier, P. y Penadés M. C. (2008). Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP). Universidad Politécnica de Valencia. https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Metodolog%C3%ADas%20%C3%81giles%20-%20Extreme%20Programming.doc (última consulta 25/03/2010)
- López Nieva, A. (2009). Innovación en la Gestión de Proyectos. Una historia en evolución. http://www.noticias.com/gestion-tecnologia-ingenieria-evolucion-infraestructuras-telecomunicaciones-tn2.2702 (última consulta 18/03/2010).
- Macchia, N. (2000). Mercadeo Relacional: Clientes para toda la vida. Debates IESA. Vol. V. No. 4.
- Mattelart, Armad (1998), Mundialización de la Información, Piados, Barcelona, 127 p.
- M. E. Johnson-Cramer, S. Parise, R. L. Cross (2007). "Managing Change through Networks and Values (Gestión del cambio a través de redes y valores). California Management Review, Vol. 49, No. 3.
- Miaton, L. (2008). Practicas ágiles en contextos distribuidos (Parte 1). Fuente Explorador Internet: http://miaton.blogspot.com/2008/02/practicas-giles-en-contextos_24.html (última consulta 02/03/2010)
- Miaton, L. (2008). Practicas ágiles en contextos distribuidos (Parte 2). Fuente Explorador Internet: http://miaton.blogspot.com/2008/02/practicas-giles-en-contextos.html (última consulta 27/03/2010)
- Mortice Kern Systems Inc. (MKS). The Compelling Need for Web-centric Software Configuration Management for Distributed Development. White Paper.
- Navegapolis (2009). Herramientas de uso libre para gestión ágil de proyectos. Fuente Explorador Internet: http://www.navegapolis.net/content/view/902/85/ (última consulta 10/05/2010)
- Navegapolis (2009). Herramientas de uso libre para gestión de proyectos. Fuente Explorador Internet: http://www.navegapolis.net/content/view/56/49/ (última consulta 05/04/2010)
- Nicolosi, Alejandra (2006). Parte I: Teletrabajo: Una mirada Global. Parte II: El caso argentino. Informe sobre Teletrabajo. Universidad Nacional de Quilmes, Argentina.
- Nilles, Jack (1976). Telecommunications-Transportation Tradeoff: Options for Tomorrow, John Wiley & Sons, Inc., New York.
- Nora, S. y Minc, A. (1980), La informatización de la sociedad, Fondo de Cultura Económica, México, 244p.

- ONGEI, Oficina Nacional de Gobierno Electrónico e Informática (1997). Herramientas para el desarrollo de Sistemas de Información. http://www.ongei.gob.pe/publica/metodologias/Lib5083/INDEX.HTM (última consulta 03/06/2010).
- Poppendieck M., Poppendieck T. (2003). Lean Software Development: An Agile Toolkit for Software Development Managers. Addison Wesley.
- Pressman, R (2002). Ingeniería del software. Un enfoque práctico. 5a. ed. Madrid: McGraw- Hill.
- Rancan, C. (2003). Gestión de Configuración de productos software en etapa de desarrollo. Trabajo final de la Especialidad en Control y Gestión de Software. ITBA.
- Raymond, E. (1998) "La Catedral y el Bazar", http://es.tldp.org/Otros/catedral-bazar/cathedral-es-paper-00.html (última consulta 10/06/2009)
- Ribagorda y Ramos (1999). Universidad Carlos III de Madrid, Madrid.
- Rivera, Marcelo (2003). Herramientas CASE en el desarrollo de Sistemas. Universidad de Concepción. Facultad de Ingeniería. Trabajo Práctico.
- Sánchez Mendoza, María (2004). Metodologías de desarrollo de software. Microsoft Certified Professional. TeamSoft.
- Schwaber K., Beedle M., Martin R.C. (2001). Agile Software Development with SCRUM. Prentice Hall.
- Skyrme, D. J. (1999). Getting to Grips with Groupware. Fuente Explorador Internet: http://www.skyrme.com/insights/7gw.htm (última consulta 19/06/2009)
- Stapleton J. (1997). Dsdm Dynamic Systems Development Method: The Method in Practice. Addison-Wesley.
- S. Parise, J. C. Henderson. (2001). "Knowledge Resource Exchange in Strategic Alliances (El intercambio de recursos de conocimientos en las alianzas estratégicas)." IBM Systems Journal, Vol. 40, Número 4.
- S. Thevenin. (2005) "Collaboration in Action: An IBM Collaboration Framework to enable Performance, Innovation and Growth (La colaboración en acción: marco de colaboración de IBM para permitir el desempeño, la innovación y el crecimiento)" IBM On Demand Learning, IBM Press, v.04a.
- White, Brian. (2000). Software configuration management. Strategies and Rational Clear Case: A practical introduction.
- Wolf, Alexander L. Van der Hoek, André and Heimbigner, Dennis. Department of Computer Science University of Colorado (1995). A Generic, Peer-to-Peer Repository for Distributed Configuration Management.

8. ANEXO

Gran parte de la motivación en la realización del presente trabajo, surge como consecuencia de haber leído un artículo de Martin Fowler [2006], “Using an Agile Software Process with Offshore Development”. Dicho artículo fue la piedra fundamental que me permitió elaborar el presente trabajo. Este artículo posee una visión detallada de la problemática de desarrollar software con equipos de trabajo dispersos geográficamente. Por otro lado también ayudó a encontrar una solución a diferentes aspectos que hay que tener en cuenta a la hora de implantar una metodología de desarrollo de software con equipos de trabajos dispersos geográficamente.

A continuación se transcriben las partes más importante del artículo en cuestión:

Utilizando un proceso de desarrollo ágil en un proyecto offshore

Durante los últimos cuatro años ThoughtWorks ha operado un laboratorio en Bangalore, India para apoyar nuestros proyectos de desarrollo de software en América del Norte y Europa. Los enfoques tradicionales de desarrollo offshore se basan en metodologías enfocadas en la planificación, pero estamos muy firmes en el campo ágil. Aquí discutimos nuestras experiencias y lecciones aprendidas en el desarrollo de software mediante la metodología ágil. Hasta ahora hemos descubierto que podemos hacer que funcione, aunque los beneficios son aún objeto de debate.

Uno de los principios fundamentales de una metodología de software ágil es la importancia de la comunicación entre las diversas personas involucradas en el desarrollo de software. Además, los métodos ágiles ponen un gran énfasis en la comunicación cara a cara entre sus integrantes. Como dice el manifiesto ágil "un método es eficiente y eficaz cuando se transmite información a un equipo de desarrollo mediante una conversación cara a cara". XP hace hincapié en esto con su práctica, donde el equipo puede trabajar en estrecha colaboración. El libro de Cockburn pasa mucho tiempo hablando acerca de la importancia de la proximidad física en los métodos ágiles.

Otra tendencia que ha estado tomando el mundo del desarrollo de software es moverse hacia el desarrollo offshore, donde gran parte del trabajo de desarrollo se hace con remuneraciones más bajas. El desarrollo offshore parece oponerse al desarrollo ágil en un par de cuestiones.

Para empezar inmediatamente va en contra de la noción de proximidad física, ya que por definición los desarrolladores offshore están muy lejos.

En segundo lugar las organizaciones offshore están a favor del enfoque basado en la planificación donde los requisitos detallados o diseños son enviados a los equipos de trabajo offshore para que sean construidos.

Entonces la pregunta fundamental es si las técnicas ágiles se pueden utilizar en un proyecto offshore. Si la respuesta es sí, entonces, ¿cómo se compara a la utilización de una metodología basada en la planificación?

Las experiencias que estoy escribiendo aquí se basan en trabajos realizados en los últimos años en ThoughtWorks. Abrimos una oficina en Bangalore, India en 2001 y se han hecho varios proyectos utilizando un equipo de trabajo en Bangalore. También hemos hecho un desarrollo offshore en nuestras oficinas en Australia. En estos proyectos nos hemos comprometido a utilizar un enfoque ágil, ya que creemos que la agilidad es un enfoque que les interesa a nuestros clientes. En este ensayo voy a describir algunas de las lecciones que hemos aprendido hasta ahora.

Para ayudar a proporcionar algún contexto, vale la pena hablar un poco sobre la forma en que hemos configurado la oficina de Bangalore. La oficina se utilizó principalmente como un entorno de desarrollo offshore por lo que en su mayoría se reclutó desarrolladores de aplicaciones. La mayoría de los contratados eran egresados de las universidades. Otro tanto eran programadores experimentados. Como resultado tenemos un grupo muy talentoso de programadores con una mezcla de niveles de experiencia. Al principio se incorporaron al proyecto experimentados desarrolladores de EE.UU. y del Reino Unido, para acompañar a los desarrolladores con menos experiencia en el proceso de desarrollo de software y en el entendimiento de las prácticas XP.

Actualmente tenemos alrededor de 150 desarrolladores en Bangalore.

Lecciones aprendidas

Uso de la integración contínua para evitar dolores de cabeza

He escuchado varias historias acerca de los problemas con la integración de los trabajos con equipos multi sitios.

Incluso con mucho cuidado en la definición de interfaces, los problemas de integración existen. A menudo esto se debe a que es muy difícil especificar correctamente la semántica de una interfaz, por lo que incluso si usted tiene todo definido, podrá aún tropezarse en asumir lo que la implementación hará.

La primera práctica de XP realizadas en ThoughtWorks fue la integración continua, y nos hemos acostumbrado tanto a ella que estábamos decididos a usarlo con nuestro desarrollo

offshore. Así que desde el principio hemos puesto todo el código en una única base, con CruiseControl ejecutándose a fin de generar y ejecutar pruebas automáticamente.

De esta manera todos se mantienen cerca de la línea principal, independientemente de su localización geográfica. Todo el mundo ha quedado encantado con lo bien que funciona esto. Su principal beneficio es que los problemas que afectan a otros grupos con la integración no nos pasan a nosotros.

La integración continua y el proceso de prueba se resuelven muchos problemas de integración muy rápidamente, como consecuencia, se puede arreglar antes de que sean difíciles de encontrar. En la página web de CruiseControl permite a todos los sitios ver lo que está pasando en otras partes.

A primera hora de la mañana se puede consultar la página web de CruiseControl para ver qué cambios se han hecho en los otros sitios. Proporciona una forma fácil de ponerse al día sobre lo que está pasando en el otro extremo del mundo. Esto requiere construir una disciplina, donde los desarrolladores se esfuerzan no romper lo construido, y solucionar de inmediato si éste se rompe. Es una práctica aceptada de que si se confirman los cambios en la codificación, los programadores afectados no deberían regresar a sus casas hasta que hayan recibido el mensaje de correo electrónico de CruiseControl que dice que los cambios efectuados resultaron en una construcción exitosa.

Una mala noche de construcción es mucho más grave cuando la oficina remota está ejecutando la misma construcción. Aunque la integración continua es muy popular en el mundo, nos hemos encontrado con algunos problemas. Las líneas de comunicación no son tan amplias y confiables como quisiéramos, así que muchas operaciones de control de código fuente pueden volverse complicada desde un sitio remoto.

En general mantenemos la construcción en servidores en el mismo sitio donde está la mayoría de los desarrolladores, pudiendo los sitios remotos demorar un largo tiempo para obtener una nueva actualización del software. Cuanto más extenso son las líneas de comunicación, más propensos a que ocurra cualquier cosa, desde problemas técnicos de las líneas a estar caídas por un tiempo. Tener el repositorio accesible las 24 horas hace complicado los procesos de resguardo de los datos.

Todas estas cuestiones podrían ser mitigadas por un repositorio de código en clúster, pero no hemos experimentado con nada de eso todavía. Hemos encontrado que algunos sistemas de control de código fuente son particularmente propensos a largas comprobaciones, sobre todo si no se han configurado correctamente.

Es penoso gastar tiempo en encontrar formas de acelerar los tiempos de comprobación y, de hecho, cambiar los sistemas de control de código fuente si uno se encuentra con uno que no lo maneja adecuadamente de manera remota.

Es interesante ver como la gente asume que estos problemas de comunicación son específicamente un problema con un sitio remoto como la India – pero hemos encontrado que a menudo se presentan problemas de infraestructura en la cede central. La integración continua requiere de una buena conectividad e incluso una mejor conectividad que la gente está acostumbrada. Incluso con la mejor conectividad, sin embargo, puede haber problemas si un sistema requiere de servicios que están detrás de un firewall. Tendrá que construir buenas pruebas dobles para estos servicios ya que el sistema pueda ser efectivamente probado en el entorno de desarrollo.

Enviar embajadores de un sitio a otro

Como he dicho anteriormente, los métodos ágiles hacen hincapié en la importancia de la interacción humana cara a cara. Incluso si todos no pueden estar colocalizados, mover algunas personas ayuda mucho. Desde el principio decidimos garantizar que en todo momento debía haber alguien del actual equipo de EE.UU. en la India para facilitar la comunicación.

Este embajador conoce a la gente establecida en EE.UU. y agrega a sus contactos personales para ayudar a que todos se comuniquen. Hemos ampliado esto a varios niveles. Hemos encontrado que es útil enviar un desarrollador y un analista de EE.UU. a la India para comunicar los aspectos técnicos y funcionales. También es valioso enviar a alguien de la India con el equipo de EE.UU. Uno de los beneficios del embajador orientado al negocio es que ayuda a proporcionar cierta información del negocio al equipo offshore.

Al construir software se pierde el contexto del negocio – a los desarrolladores se les dice qué hacer sin que se les digan por qué es importante lo que hacen. El por qué a menudo hace una gran diferencia en hacer las cosas apropiadamente. Una parte importante del trabajo del embajador es comunicar lo que se habla. En cualquier proyecto hay mucha comunicación informal. Así que parte del trabajo de los embajadores es comunicar un montón de cosas que no parecen lo suficientemente importantes como para obtener más canales de comunicación formal.

Por lo general, los embajadores rotan cada determinado tiempo, ya que si un embajador está mucho tiempo en el extranjero pierde el contacto local. Esto hace que sea más fácil para los embajadores, quien no desea estar mucho tiempo afuera. En la elección de embajadores es muy importante prestar atención a las necesidades y preferencias individuales. Algunas

personas no quieren estar lejos de sus hogares, por lo tanto estas personas no deben ser embajadores.

También hemos encontrado que es importante que los gerentes de proyecto utilicen parte de su tiempo como embajadores. Gran parte del trabajo del gerente de proyecto es ayudar a resolver conflictos y solucionar problemas antes de que se agraven. Los embajadores son los encargados de construir confianza. Como resultado de ello es esencial enviar embajadores lo antes posible en el proyecto. Si un proyecto se ejecuta sin embajadores, los problemas de comunicación llevarán mucho trabajo en solucionarlos. Los embajadores reducen este riesgo, pero es necesario que estén desde el principio antes que los problemas comiencen a aparecer.

Realice visitas seguidas para generar confianza

Los embajadores son personas semi-permanentes que pasan varios meses en la "otra" locación. Esto es vital, pero no suficiente. También necesita hacer más contactos que implican un mayor número de personas.

Existen dos tipos de visitas: las visitas que se producen al principio del proyecto y están destinadas a crear las relaciones, manteniendo al mismo tiempo visitas para ayudar a que las relaciones sigan su camino. Las visitas deben ser planificadas al principio del proyecto y deben ser sustanciales en tiempo – mínimo dos semanas. Enviar a algunos clientes onshore y directores de proyectos al sitio offshore para crear un plan de lanzamiento inicial.

- Haga que los analistas offshore vayan al sitio onshore para que sean parte de las sesiones de análisis de requerimientos.
- Haga que algunos desarrolladores onshore visiten al equipo offshore para que trabajen en las primeras iteraciones
- Haga que el equipo offshore visite el sitio onshore - sobre todo si en el sitio onshore se encuentra el cliente

Cualquiera sea la razón, recuerde que el propósito principal de la visita es la construcción de las relaciones de trabajo. Las relaciones informales son de gran importancia para construir confianza.

No subestime el Cambio Cultural

Una de las partes más duras de la introducción de métodos ágiles en una organización es el cambio cultural que provoca. De hecho hemos encontrado que esta es la razón principal por la que las organizaciones tienen problemas con la adopción de métodos ágiles. Muchas empresas operan con un modelo de control de mando que asume que los niveles senior toman decisiones y los de menos jerarquía llevan a cabo las tareas.

Para que los métodos ágiles trabajen se necesita mucho más autonomía y la toma de decisiones de los hacedores.

Nos parece que esto es un gran problema en las sociedades occidentales, pero el problema se amplifica en Asia ya que las culturas asiáticas refuerzan las deferencias jerárquicas. En este entorno las personas se desalientan a menudo de hacer preguntas, hablar de los problemas, advirtiendo acerca de los plazos inviable, o proponer alternativas a las instrucciones percibida a partir de superiores.

La mala noticia es que cuesta muchos que los equipos lleguen a ser más preactivos e inevitablemente lleva mucho tiempo. Hacer que la gente utilice un estilo de control distribuido de la gestión requiere más tiempo de lo que se piensa.

Pero hay buenas noticias. Una vez que las personas son conscientes de la libertad, y la responsabilidad de tomar decisiones - realmente lo disfrutan.

Varias personas del equipo en India han dicho que sus amigos en otras compañías no pueden creer todo lo que se les da de autonomía. Esta autonomía es un gran motivador, permitiendo a la gente que sea más productiva y capaces de crecer en una mayor responsabilidad. Para mí una de las cosas más interesantes que se descubren es lo que los efectos a largo plazo son de este impacto cultural, tanto en Asia como en Occidente. Las personas tienen muchas más probabilidades de plantear cuestiones si tienen una buena relación personal.

(En ThoughtWorks tenemos una actitud anti-autoridad que es sorprendente, incluso en los EE.UU. Decidimos desde el principio que íbamos a mantener esa misma cultura de la India. Me alegra decir que sin duda parece estar teniendo éxito.)

Use wikis para compartir información en común

Hemos jugado de diferentes maneras para compartir información en común, nuestro favorito hasta ahora es el wiki. Los wikis funcionan bien porque son fáciles de usar, se puede trabajar con cualquier navegador y son fáciles de configurar.

Cualquier información puede ser puesta ahí, las tarjetas de historia, guías de diseño, instrucciones de compilación, notas sobre los progresos realizados - cualquier cosa que necesita ser escrito por el equipo. Hemos encontrado que es muy útil la capacidad de notificación de cambio que tienen muchos wikis.

Los wikis son por naturaleza no estructurada, y esta falta de estructura es parte de la prestación. El equipo usualmente puede desarrollar su propia estructura en el wiki a medida que el proyecto crece.

Utilice scripts de pruebas como ayuda para entender los requerimientos

La comunicación es importante a la hora de obtener los requerimientos. He encontrado equipos XP maduros que utilizan pruebas de aceptación como medios de comunicación para obtener los requerimientos. Estos equipos escriben los scripts de prueba antes del inicio de una iteración para ayudar a aclarar las necesidades y le dan al equipo de desarrollo un objetivo concreto para alcanzar la meta. El objetivo es escribir un relato corto (un par de páginas) para completar una función (la historia en XP). Un analista/tester basado en la India entonces crea un script de prueba para esa historia. Esto puede hacerse con un proceso automático o manualmente, aunque muchos prefieren pruebas automáticas.

Los scripts son desarrollados en EEUU y los analistas de la India coordinan por email o chat conferencias regulares (2-3 veces por semana) a fin de revisar esos scripts de pruebas. Hemos encontrado que esto ha ayudado mucho al analista indio y al cliente de los EE.UU. entender los requerimientos. Las pruebas obligan al analista indio a entender realmente lo que se necesita.

A los desarrolladores les resulta más fácil hacerle preguntas al analista de la India en lugar de excavar a través de scripts de prueba, así que tener un analista/tester indio sigue siendo importante. Los motores de búsqueda son buenos, pero los seres humanos suelen ser más fáciles de trabajar. El mayor problema que encontramos con esta técnica es involucrar al personal del cliente.

Utilice construcciones regulares para obtener información sobre las funcionalidades

Cuando la gente considera la recopilación de requisitos en los métodos ágiles, a menudo no ven la importancia de la retroalimentación. A menudo, el proceso de requisitos es visto como el trabajo de los analistas obteniendo los requisitos y los desarrolladores programando e implementando. En un proyecto ágil, la cercanía entre el cliente y el desarrollador permite al cliente controlar el progreso con mucha más frecuencia, lo que les permite detectar malos entendidos con mayor rapidez.

Además mostrarle un sistema parcialmente desarrollado puede educar al cliente, pues a menudo hay una diferencia entre lo que pide y lo que necesitaba. A medida que se realizan las construcciones de integración, los clientes de EEUU esperan los resultados. Si bien esto no es tan inmediato como la coubicación, aún permite al cliente corregir cualquier malentendido con rapidez, incluso les permite redefinir su propio entendimiento de los requerimientos. Para que esto funcione, es fundamental solucionar los problemas de ambiente, duplicando los ambientes onshore y offshore. No hay nada peor que la gente onshore encuentre un problema, y las personas offshore no puedan reduplicar el problema debido a problemas de configuración del entorno.

Asegúrese de que el entorno esté ordenado desde el inicio y asegurase que alguien esté capacitado para solucionar cualquier problema que se presente en el ambiente. Asegúrese de que la gente se concentre en hacer las construcciones regularmente, incluso si es sólo

funcionalidad parcial. Scrum ha defendido que el equipo de desarrollo hace una demostración a los clientes al final de cada iteración. Hemos adoptado esta práctica muy ampliamente ahora. Con los equipos remotos nos gusta hacer una demostración, donde los desarrolladores offshore muestran las nuevas características del software con la ayuda de la aplicación de escritorio remoto.

Utilice reuniones de estado cortas

Los métodos ágiles promueven periódicamente reuniones de estado cortas para todo el equipo. Llevar esta actividad a los grupos remotos es importante para poder coordinar con otros equipos. Las zonas horarias son a menudo el mayor problema aquí, sobre todo entre los EE.UU. y la India. En nuestros proyectos anteriores encontramos que dos veces por semana parecía funcionar bien.

En nuestros proyectos más recientes hemos utilizado wikis lo que ha reducido la brecha comunicacional. No obstante hacemos reuniones diarias con los integrantes onshore y offshore, pero éstas no tienen que ver con todo el equipo - sólo se involucran aquellos que se centran en la colaboración onshore y offshore. Hemos encontrado que es una buena costumbre iniciar las teleconferencias con novedades del ámbito local. La idea es no perder de vista el contexto en el que vivimos.

Con los problemas de huso horario es importante para ambas partes escoger el momento para hacer las llamadas. Uno de nuestros clientes sólo haría las llamadas durante su día laboral, lo que obligó a la gente de la India a estar en horas difíciles. Planificamos el lanzamiento de un proyecto durante la fiesta india de Diwali, en este caso es el equivalente a pedirle a un equipo de EEUU que trabaje el día de Acción de Gracias. Afortunadamente pudimos convencerlos de que cambien la fecha.

Utilice iteraciones cortas

En general los métodos ágiles utilizan iteraciones más cortas que muchos otros enfoques iterativos. En casi todos los proyectos de ThoughtWorks se utilizan iteraciones de una o dos semanas de duración. Algunos de los desarrolladores indios más experimentados han trabajado en lugares donde utilizan de dos a tres iteraciones por mes y afirman que las iteraciones más cortas son mucho mejores.

Un par de años atrás nuestros proyectos distribuidos habitualmente realizaban dos iteraciones por semana ya que descubrieron que era difícil utilizar iteraciones más cortas, pero esto ha cambiado. Ahora hemos aprendido a hacer una iteración por semana en un proyecto distribuido.

Utilice un sistema de reuniones de planificación de iteración que se adapte para sitios remotos

En la mayoría de nuestros proyectos, hemos descubierto que una reunión de planificación al principio de cada iteración, que involucre a todo el equipo, realmente ayuda a conseguir

que todos estén coordinados en la siguiente etapa del trabajo. Me he dado cuenta que muchos de nuestros proyectos han utilizado su propia variante de Reuniones de Planificación de Iteraciones (RPI) para adaptarse a las circunstancias locales. (Este tipo de auto-adaptación es una parte importante de los procesos ágiles.) . Un equipo remoto agrega su propio conjunto de limitaciones, especialmente cuando se tienen problemas de zona horaria.

Antes de la RPI, los clientes de los EE.UU., envían narrativas de cada funcionalidad (historia) las que transformamos en scripts de pruebas. Durante este período algunas preguntas se tratan por correo electrónico. Justo antes de la RPI el equipo de desarrollo "rompe" las características funcionales en tareas más detalladas. Este desglose de las tareas son compartidas con los EE.UU. Todo esto es el trabajo previo a la llamada telefónica. Vimos que las llamadas telefónicas suelen durar desde media hora hasta un par de horas.

La corrección de errores es un buen comienzo para el equipo offshore

Dos de nuestros proyectos consiste en tomar un gran tamaño (cientos de miles de líneas de código) de código base y trasladar el desarrollo sustancial del código base al laboratorio de Bangalore. En ambos proyectos el equipo indio comenzó con unas pocas iteraciones de corrección de errores antes de comenzar a agregar nuevas funcionalidades.

Al comenzar con las correcciones de errores, le permitió al equipo indio familiarizarse con el código base antes de que ellos hicieran el trabajo sustancial. Aunque esto funciona bien, hay una cierta preocupación que las personas más experimentadas puedan considerar que es un estigma hacer únicamente correcciones de errores. Si bien algunas personas pueden percibir esto como un problema creo que trabajando en las correcciones de errores o sobre cambios de características es una de las mejores maneras de familiarizarse con un nuevo código base.

La naturaleza de la comunicación en la corrección de errores también puede hacer que sea una actividad offshore más fácil. Los equipos onshore pueden pasar su día mapeando los detalles de los errores, que pueden ser comunicados al equipo offshore para que puedan trabajar durante la noche del sitio onshore. El equipo onshore puede revisar las correcciones al día siguiente. Debo subrayar que esto no es más eficiente que tener un equipo in situ corrigiendo los errores, debido a la dificultad de comunicación, pero puede ser una forma menos complicada de trabajar con un equipo offshore.

Separar los equipos por funcionalidad, no por actividad

Gran parte del pensamiento tradicional sobre los límites onshore/offshore se basa en la actividad que la gente hace. Así pues el análisis y el diseño se hacen onshore, la construcción se realiza offshore, y las pruebas de aceptación se ejecutan onshore. Obviamente, esto encaja bien con el modelo de cascada.

Hemos descubierto que, contrariamente a esto, las cuestiones mejoran cuando hacemos que el equipo offshore maneje tantas actividades como sea posible. Por lo tanto preferimos verlos hacer mucho trabajo de análisis y diseño como sea posible, sujeto a la limitación que los requisitos son procedentes del sitio onshore. Cuando dividimos a los equipos de desarrollo offshore y onshore, lo hacemos de acuerdo a los argumentos de funcionalidad y no de actividades. Rompemos el sistema en módulos y dejamos que el equipo offshore tome algunos de estos módulos. Sin embargo a diferencia de la mayoría de los grupos, no hacemos un gran esfuerzo para diseñar y congelar las interfaces entre los módulos: la integración continua permite que los módulos de interfaces evolucionen a medida que el desarrollo continúa.

Una parte importante de esto es hacer crecer la parte analista del equipo offshore. Los desarrolladores entienden el negocio, además el equipo de desarrollo puede trabajar de manera eficaz. En lugar de tener que esperar toda la noche para responder a una pregunta, los desarrolladores pueden obtener respuestas de inmediato - pueden eliminar bloques para progresar. Todo esto significa que usted tiene que centrarse en hacer crecer el conocimiento del negocio de los analistas offshore. Esto toma tiempo, pero el conocimiento local es un complemento indispensable para conocer el negocio onshore. Un corolario de esto es no dividir equipos en sentido horizontal (por ej., tener un equipo que haga la capa de presentación y otro que haga la capa de persistencia). En general, yo prefiero una organización con personal funcional.

La cosa más importante para recordar aquí es el poder dominante de la Ley de Conway - la estructura del sistema reflejará la estructura del equipo que lo construyó. Eso significa que es importante separar los equipos distribuidos por módulos que se encuentren tan acoplados como sea posible.

Esperar a necesitar más de los documentos

Los métodos ágiles restan importancia a la documentación a partir de la observación de que una gran parte del trabajo de documentación se pierde. La documentación, sin embargo, se vuelve más importante con el desarrollo offshore ya que la comunicación cara a cara se reduce. Este trabajo es, en cierto modo, un desperdicio, ya que no sería necesario si todo el equipo fuera coestablecido.

Además de documentos, usted pude tener la necesidad de utilizar herramientas colaborativas: wikis. Aconsejo siempre a los equipos comprobar sus documentos en un sistema de control de versión así la gente puede conseguir fácilmente el material actualizado. Esto es particularmente importante cuando usted está haciendo el trabajo remoto.

Obtener varios modos de comunicación para trabajar rápido

Diferentes herramientas de comunicación de trabajo para diferentes tipos de problemas. Como mínimo asegúrate de tener un wiki, mensajería instantánea, teléfono y una buena conexión. El correo electrónico a menudo puede ser una bendición a medias. En particular, hemos encontrado que es bueno desalentar el correo electrónico persona a persona en favor de grupos de noticias o listas de correo. Mediante la publicación de mensajes y peticiones en un grupo de noticias, todos pueden ver los mensajes y es fácil de buscar. La gente encuentra más fácil pasar por alto temas que no les interesa. Algunos equipos han reportado efectos positivos al usar Campfire (el IRC podría ser utilizado de una manera similar, aunque no he oído mencionar todavía.) .

Quizás lo más complejo de resolver es cómo comunicar la “imagen grande” - la visión del proyecto. La mayor parte de la comunicación, y la discusión acerca de la comunicación, se concentra en los detalles del día a día. Es importante obtener estos derechos, pero hay peligro de que centrarse en el día a día nadie presta atención a la visión total del proyecto. Esto puede ser perjudicial porque muchas personas se habitúan a realizar pequeñas decisiones basadas en su percepción de la gran visión. Esta cuestión es particularmente importante para comunicar el contexto del negocio de un proyecto. La comunicación remota a menudo se centra demasiado en los detalles tácticos – que se necesita ser construido esta semana. Pero muchas decisiones técnicas necesitan un contexto estratégico más amplio - es importante que el equipo remoto tenga una visión más amplia de la dirección el proyecto.

Esta clase de comunicación está careciendo a menudo en proyectos in situ, particularmente cuando hay muchas barreras entre el negocio y la tecnología. Los equipos de desarrollo remotos exacerban este problema – al igual que el desarrollo distribuido exacerba la mayoría de las dificultades de la comunicación.