Diseño de una Metodología Ágil de Desarrollo de Software.

Schenone Marcelo Hernán.

mscheno@fi.uba.ar

Tesis de Grado en Ingeniería en Informática.

Facultad de Ingeniería.

Universidad de Buenos Aires.

- 2004 -

Abstract

Esta tesis tiene como propósito la construcción de una Metodología Ágil de Desarrollo de Software la cual utiliza UML como notación. Si bien podrá ser empleada en proyectos de distinto tamaño y complejidad, su aplicación tendrá como objetivo proyectos de pequeña escala y riesgo limitado. También será independiente del lenguaje o la arquitectura utilizada, así como del tipo de software que se está construyendo.

Para desarrollar esta metodología se comenzará por un relevamiento de las metodologías y notaciones actualmente empleadas (Rational Unified Process, UML, SCRUM, OPEN, Extreme Programming, etc), un posterior refinamiento de las mismas y el desarrollo paulatino de un proceso que incorpore las mejores y más avanzadas prácticas existentes en cada etapa del desarrollo.

Finalmente, se describe la realización de dos casos prácticos resueltos con la metodología propuesta. El primer caso práctico estará basado en un sistema de integración de servicios para ONGs, y el segundo en un sistema de administración de recursos de hardware y software.

Tabla de Contenidos

Tabla de Contenidos Detallada

G. Capítulo III - Solución Propuesta
    a. Descripción de Aspectos a Incluir en una Metodología Ágil
        i. Características de la Metodología
        ii. Roles
        iii. Fases e Hitos
            1. Concepción
            2. Elaboración
            3. Construcción
            4. Transición
        iv. Disciplinas dentro de las Fases
            1. Factibilidad
            2. Requerimientos – Análisis
            3. Diseño
            4. Implementación – Testing
            5. Despliegue
        v. Disciplinas de Soporte
            1. Administración de Proyecto
            2. Administración de la Configuración
            3. Administración del Proceso
            4. Administración de Personas
            5. Administración del Conocimiento
        vi. Artefactos
            1. Visión
            2. Plan de Proyecto
            3. Lista de Riesgos
            4. Modelo de Casos de Uso y Especificaciones de Casos de Uso
            5. Documento de Especificación de Requerimientos de Software (SRS)
            6. Descripción de la Arquitectura
            7. Casos de Prueba
            8. Scripts de Despliegue
            9. Planilla de Incidentes
            10. Repositorio del Proyecto

Prefacio

Esta Tesis de Grado fue realizada durante el período que va desde Agosto de 2001 hasta Abril de 2004.

La elección del tema surgió a partir de la investigación del autor en relación a los Modelos de Proceso de Desarrollo observados en diversas materias de la carrera. A medida que comenzaba la exploración llegó a su conocimiento la metodología eXtreme Programming (XP) desarrollada por Kent Beck a las cuales se incorporaron muchas cuales que conformarían el universo de metodologías ágiles. Dado el énfasis de las mismas en cuestiones de Peopleware, Dinámica de Equipos, Psicología Social, Calidad del Proceso, el tema fue elegido como base para la construcción de una metodología que analizara estos aspectos, basándose en las mejores practicas de la industria e incorporando aspectos interdisciplinarios tomados de la Psicología, Sociología, Relaciones de Trabajo y Administración. La idea de la misma era que pudiera ser utilizada dentro de la realidad de la informática en nuestro país. Así nace la presente tesis que describe en detalle AgEnD, la metodología ágil que es motivo del presente trabajo.

A fines del 2000, manteniendo conversaciones con el docente Sergio Villagra – en ese momento jefe de trabajos prácticos de la cátedra de Proyectos Informáticos I de la FIUBA – quien mostró gran interés en el tópico elegido, se planteó el ofrecimiento de éste de trabajar como Tutor de la Tesis. Mediante la formalización del trámite de inicio de la Tesis, comenzó el arduo desarrollo que finalizaría en el año 2004.

Organización de la Tesis

La Tesis está organizada de la siguiente forma.

- Prefacio, contiene información relevante a los orígenes y el desarrollo del trabajo
- Introducción, describe brevemente los temas a los que se hará referencia durante el trabajo
- Descripción del Problema, describe sintéticamente la necesidad de contar con un proceso de desarrollo y como las Metodologías Ágiles

intentan resolver distintos problemas surgidos de la complejidad inherente al software

- Solución Propuesta, describe detalladamente la Metodología propuesta, AgEnD, destacando los valores propuestos y analizándola en relación al universo de Metodologías Ágiles
- Resultados Experimentales, narra las experiencias realizadas con las diversas prácticas propuestas en AgEnD, junto con los resultados derivados de estas
- Conclusiones, plantea las conclusiones obtenidas del desarrollo así como futuras líneas de investigación a seguir
- Anexos, detallan temas que si bien no son centrales para la Tesis, sirven para profundizar en aspectos mencionados en esta
- Bibliografía, contiene la totalidad del material utilizado para confeccionar el trabajo

Agradecimientos

Quisiera agradecer a las siguientes personas que han contribuido de una u otra forma durante la elaboración de la Tesis. Mis agradecimientos son para: Aníbal Calligo, Germán Boccoleri, Alejandro Oliveros, Elvio Nabot, Enrique Vetere, Ángel Pérez Puletti, Adriana Yechúa, Pablo Sametband, Adrián Lasso, Pablo Cosso. A mi Tutor, Sergio Villagra.

Finalmente quisiera agradecer a mis padres, Olga y Jorge, quienes me ayudaron incondicionalmente a que pudiera completar la misma. Asimismo hago extensivo el agradecimiento a toda mi familia y demás personas que olvido en este momento mencionar.

Capítulo I - Introducción

Propósito detrás de esta Tesis

Para empezar a hablar de Ingeniería de Software daremos una definición aceptada por la industria, esta ha sido tomada de la IEEE Computer Society:

> **"(1) La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software; esto es, la aplicación de la ingeniería al software"**
>
> **"(2) El estudio de los enfoques como en (1)"**

Como podemos observar esta definición apunta a la profesionalización de la disciplina de la informática dentro del ámbito de las demás ingenierías. Al respecto debemos mencionar los constantes esfuerzos realizados por la comunidad informática para generar este cambio. El objetivo ulterior radica en que la disciplina pase de ser un conjunto de buenas prácticas y folklore a un conjunto de patrones y estándares a ser aplicados en cada situación.

Sobre la naturaleza caótica del software encontramos una importante referencia en el artículo "No Silver Bullet" de Frederick Brooks [Brooks, 1987] en el cual se analizaban las características intrínsecas del software. En el mismo, el autor tomaba la realidad de la disciplina mencionando los grandes problemas que planteaba el desarrollo de software. Las características esenciales descritas en dicho paper eran las siguientes:

- Complejidad, las entidades de software son más complejas por su tamaño que cualquier construcción humana
- Conformidad, mucha de la complejidad deviene de la necesidad de conformar con múltiples interfaces, heterogéneas entre sí ya que involucran a distintas personas
- Maleabilidad, las entidades de software están sujetas a cambios constantemente; dado que el software es materia puramente intelectual y que su cambio es percibido como algo sencillo el mismo suele sufrir este efecto

- Invisibilidad, lo que genera dificultades en la comunicación para el modelado de los sistemas

Teniendo en cuenta la bibliografía de las décadas pasadas se observa que si bien se han desarrollado diversos enfoques para mitigar la complejidad inherente al software, el mismo sigue presentando un gran desafío en su desarrollo. De ahí la conclusión de Brooks respecto a la construcción de software que citamos en forma verbatim.

> **No existe ningún desarrollo, ya sea en tecnologías como en management, que por si solo prometa siquiera una mejora de un orden de magnitud en la próxima década en lo referido a productividad, confiabilidad, simplicidad.**

Podemos decir que la predicción de Brooks se mantiene hasta el día de hoy, diecisiete años después del paper. Sin embargo podemos afirmar que se ha avanzado en muchos frentes dentro de la informática al punto que se está en camino de llegar a la profesionalización de la disciplina.

Como en todas las demás ingenierías que han tenido un mayor tiempo de maduración surge la necesidad de la estandarización. Es decir, abandonar la aplicación de técnicas y buenas prácticas en forma ad hoc a favor de un proceso disciplinado, que esté definido por normas, que resulte predecible y provea una adecuada visibilidad para los stakeholders. Estas ideas son las que impulsan el desarrollo de esta tesis.

Historia de los procesos de desarrollo

Uno de los grandes pasos dados en la industria del software fue aquel en que se plasmó el denominado modelo en cascada. Dicho modelo sirvió como base para la formulación del análisis estructurado, el cual fue uno de los precursores en este camino hacia la aplicación de prácticas estandarizadas dentro de la ingeniería de software. Propuesto por Winston Royce en un controvertido paper llamado "Managing the Development of Large Software Systems" [Royce, 1970] este modelo intentaba proponer una analogía de línea de ensamblaje de manufactura para el proceso de

desarrollo de software, queriendo forzar predictibilidad en una entidad como el software que como fue mencionado es algo complejo y que esta más relacionado con el desarrollo de nuevos productos.

Se debe tener en cuenta el momento de la historia cuando aparece este modelo. El mismo surge como respuesta al modelo codificar y probar que era el que predominaba en la década de los ‘60. En esa época ya existían modelos iterativos e incrementales pero no eran disciplinados ni estaban formalizados. A consecuencia de esta realidad, la idea de tener un modelo que ordenara el proceso de desarrollo y que parecía bastante sencillo de llevar a la práctica y de comunicar hizo que el modelo en cascada tuviera una gran promoción. Esto resulta irónico ya que como años más tarde explicó su hijo, Walker Royce, Winston Royce en realidad abocaba por los modelos iterativos y solo propuso el modelo en cascada como la descripción más simple de un proceso la cual solo serviría para proyectos muy predecibles y sin grandes riesgos [Larman, 2003].

Este modelo se basaba en el desarrollo de etapas las cuales tenían un input y un output predefinido. El proceso era desarrollado en forma de cascada ya que se requería la finalización de la etapa anterior para empezar la siguiente. Esto degeneraba en un “congelamiento” temprano de los requerimientos, los cuales al presentar cambios requerían gran esfuerzo en retrabajo. Otra opción era no permitir cambio alguno a los requerimientos una vez que se iniciara el desarrollo lo que traía aparejado que el usuario no veía la aplicación hasta que ya estaba construida y una vez que interactuaba no cubría sus necesidades.

Asimismo, dadas las características inherentes del modelo, la fase de implementación del mismo requería el desarrollo de los módulos en forma independiente con las correspondientes pruebas unitarias, y en la siguiente fase, se realizaba la integración de los mismos. Esto traía aparejados grandes inconvenientes debidos a que todo estaba probado en forma unitaria sin interacción con los demás módulos. Las sorpresas llegaban cuando se integraban estas piezas para formar la aplicación; lo cual inevitablemente desembocaba en un retraso del proyecto, sacrificando la calidad del mismo.

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

del software (subsanando muchas de las falencias del modelo en cascada) mitigando los riesgos en forma temprana. Los procesos iterativos de los cuales se desprenderían diversas instancias, como son el modelo iterativo e incremental, el modelo en espiral, el modelo basado en prototipo, el modelo SLCD, el MBASE, el RUP, etc.

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 auto-contenido el cual está compuesto por todas las fases del desarrollo (requerimientos, diseño, implementación, testing). Los incrementos están dados por la funcionalidad que se va agregando al software en forma iterativa. Gracias a estas iteraciones se logra entre otras cosas obtener el feedback necesario del cliente que era frenado en el modelo en cascada una vez se finalizaba la fase de requerimientos. Consecuentemente podemos argumentar que los modelos iterativos fomentan el cambio en forma temprana y proponen un control de cambio disciplinado que permita que el usuario ajuste sobre el desarrollo sus requerimientos. Esto se contrapone a la intolerancia del modelo en cascada para lidiar con dichos cambios.

Del modelo en espiral desarrollado por [Boehm, 1988] surgió una de las ideas fundamentales que las metodologías posteriores adoptarían: el temprano análisis de riesgos. Como muestra la Figura 001 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 relevados en el proyecto mediante la realización de prototipos o simulaciones de tipo desechables tendientes a probar algún concepto. Una vez que esos prototipos son validados se suceden iteraciones del tipo: determinar objetivos, evaluar, desarrollar, planear. Mediante estas iteraciones se procuraba el feedback del que se hablaba anteriormente, en forma temprana. Una vez que se tenía el diseño detallado y validado por el cliente, se implementaba el software siguiendo las etapas de un modelo en cascada. Esta es una falla importante del modelo ya que no se acomoda a la posibilidad de cambios una vez que se inicia la construcción. Todas las críticas que se le hacían al modelo en cascada se aplican a estas fases del modelo en espiral.

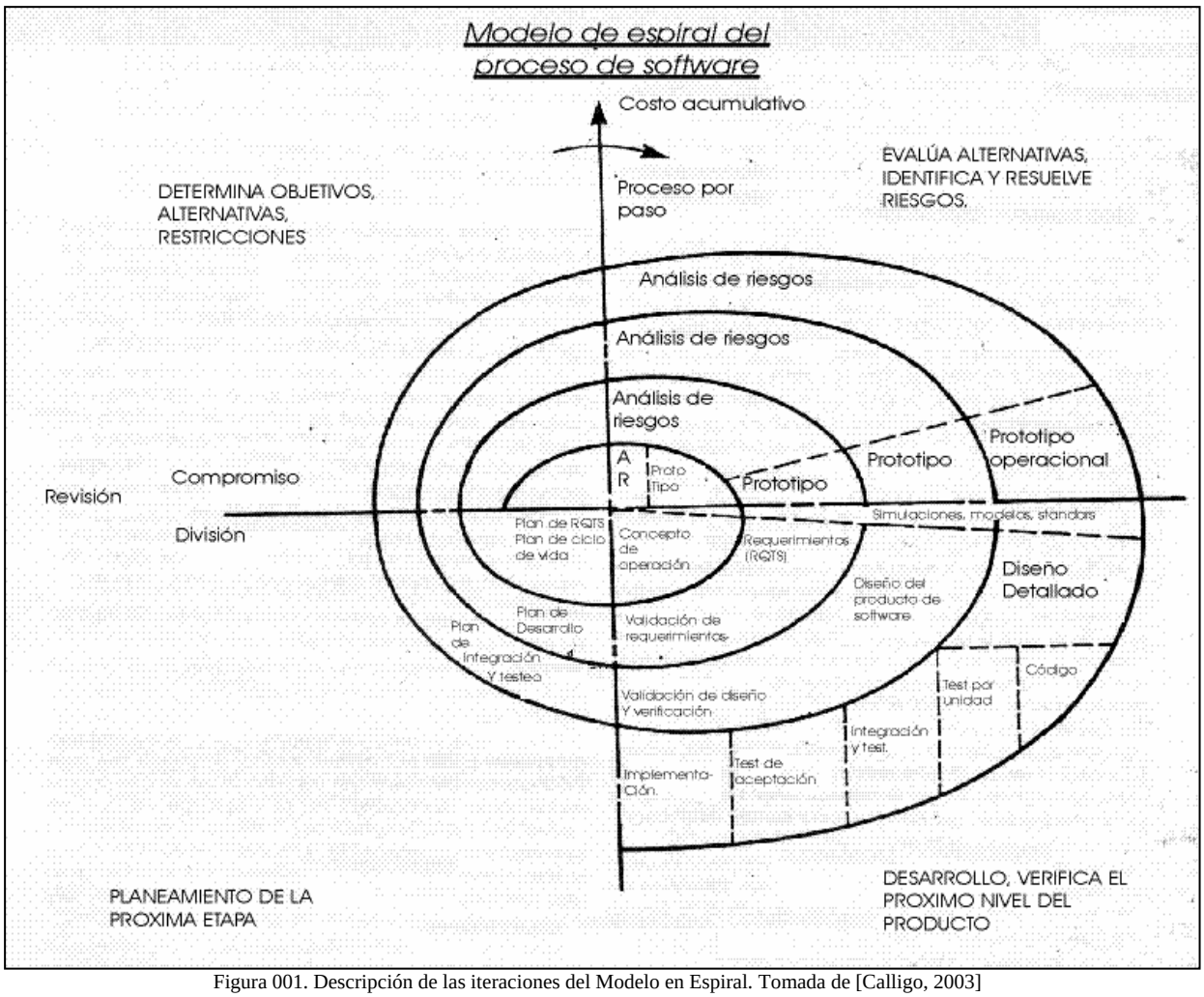

Figura 001. Descripción de las iteraciones del Modelo en Espiral. Tomada de [Calligo, 2003]

Fue el mismo Barry Boehm, autor de este modelo, quien en su artículo [Boehm, 1995] describe tres hitos críticos a ser utilizados en cualquier proyecto de forma de poder planificar y controlar el progreso del mismo, dando visibilidad a los stakeholders. Estos hitos están relacionados con las etapas de avance que se van dando a lo largo de un proyecto de acuerdo al pasaje que ocurre de las actividades de ingeniería (que componen los espirales del modelo en espiral) a las actividades de producción (que componen la construcción en cascada del software). Su impacto en la industria del software ha sido tan importante que uno de los procesos mas utilizados en la actualidad, el [RUP, 2002], los incorpora. Estos hitos son:

- Objetivos del Ciclo de Vida
- Arquitectura del Ciclo de Vida
- Capacidad Operacional Inicial

El primer hito finaliza con la definición del alcance del software a construir, la identificación de los stakeholders, y el delineamiento del plan de desarrollo del sistema. El mismo ocurre al final de la fase de *Concepción* según el RUP.

El segundo hito finaliza con el delineamiento de la arquitectura del sistema, la resolución de todos los riesgos críticos del proyecto, y el refinamiento de los objetivos y el alcance del sistema. A partir de este hito, se comienza la construcción en forma masiva del sistema, llegándose a utilizar el máximo de recursos en el proyecto. Asimismo, comienzan las fases más predecibles en cierta medida del desarrollo. El mismo corresponde al hito final de la fase de *Elaboración* según el RUP.

El último de los hitos corresponde a la entrega del primer release del software, que incorpora la funcionalidad definida en la correspondiente iteración. También se espera el tener material de entrenamiento, como un Manual del Usuario y Manual de Operaciones. Este hito se corresponde con el hito final de la fase de *Construcción* según el RUP.

Lo que resultó interesante de estos hitos propuestos por Boehm es que los mismos son independientes del proceso de desarrollo elegido. Permiten una estandarización de entregas a ser realizadas en un proyecto, la cual tiene ventajas para ambas partes comprometidas. Para los clientes, los hitos otorgan visibilidad sobre el proyecto pudiendo medir el progreso con artefactos que les son de utilidad. Para el equipo de desarrollo, los hitos proveen una guía de las fases del proyecto orientada a los entregables necesarios para cada hito así como la posibilidad de recibir feedback de los clientes/usuarios sobre los productos que son entregados en el tiempo.

Como fue mencionado en los últimos párrafos, uno de los procesos con más influencia en la comunidad del software ha sido el RUP. El mismo, derivado y refinado por Rational a partir de la absorción del Objectory de Jacobson tiene diferencias radicales en comparación con los procesos que lo han precedido. El RUP es uno de los primeros procesos que es vendido como un producto[1] y que es altamente customizable a las circunstancias en que es adoptado. La idea de los creadores del RUP es que el mismo fuera un repositorio de todas las ideas vigentes y las denominadas *Best Practices* de la Ingeniería de Software. Dada la complejidad del mismo se presenta mediante los

denominados *roadmaps* que son simplemente formas de customizar el RUP para resolver tipos de problemas específicos. Sin embargo, al intentar abarcar proyectos de envergaduras tan dispares como podrían ser la construcción de un sistema de radares para portaviones versus la construcción de un registración de usuarios para una pequeña consultora, el RUP pierde la granularidad necesaria para describir en detalle uno de los factores más trascendentes de cualquier desarrollo de software: las personas.

Esta es una de las razones principales del advenimiento de las denominadas Metodologías Ágiles.

Metodologías Ágiles de Desarrollo

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 requerida calidad. El enfoque fue planteado por primera vez por [Martin, 1991] y se dio 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. En general, se considera que este fue uno de los primeros hitos en pos de la agilidad en los procesos de desarrollo como mencionaremos a continuación. Cabe mencionar que las metodologías ágiles no inventaron la noción de los procesos iterativos e incrementales, los cuales eran usados desde décadas pasadas inclusive en momentos en que el modelo en cascada era el estándar.

La historia de las Metodologías Ágiles y su apreciación como tales en la comunidad de la ingeniería de software tiene sus inicios en la creación de una de las metodologías utilizada como arquetipo: XP - eXtreme Programming. XP surge de la mente de Kent Beck, tomando ideas recopiladas junto a Ward Cunningham, y utilizando conceptos como el de Chief Programmer creado por IBM en la década de los ’70.

Durante 1996 Beck es llamado por Chrysler como asesor del proyecto *Chrysler Comprehensive Compensation (C3) payroll system*. Dada la pobre calidad del sistema

[1] De hecho, Rational que fue adquirida por IBM en el año 2003 vende el RUP junto con templates y documentación anexa para uso comercial a empresas.

que se estaba desarrollando, Beck decide tirar todo el código y empezar de cero utilizando las prácticas que él había ido definiendo a lo largo del tiempo. El sistema, que administra la liquidación de aproximadamente 10.000 empleados, y consiste de 2.000 clases y 30.000 métodos, es puesto en operación en Mayo de 1997 casi respetando el calendario propuesto [C3, 1998] [Williams, 2000]. Como consecuencia del éxito de dicho proyecto, Kent Beck dio origen a XP iniciando el movimiento de metodologías ágiles al que se anexarían otras metodologías surgidas mucho antes que el propio Beck fuera convocado por Chrysler.

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
- Extreme Modeling

A continuación, daremos un breve resumen de dichas metodologías, su origen y sus principios. La idea es poder observar las similitudes entre las mismas y aquellas bases que unifican los criterios con los que fueron creadas y desembocaron en el Manifiesto descrito posteriormente.

XP – Extreme Programming

La primera metodología ya ha sido comentada y es la que le dio conciencia al movimiento actual de metodologías ágiles. De la mano de Kent Beck, XP ha conformado un extenso grupo de seguidores en todo el mundo, disparando una gran cantidad de libros a los que dio comienzo el mismo Beck en [Beck, 2000]. Inclusive Addison-Wesley ha creado una serie de libros denominada *The XP Series*. Fue la misma gente del proyecto C3 la que produjo también otro de los libros importantes de XP

[Jeffries, 2001] en el que se bajaban los conceptos de Beck a la puesta en práctica en un proyecto.

La imagen mental de Beck al crear XP [Beck, 2000] era la de perillas en un tablero de control. Cada perilla representaba una práctica que de su experiencia sabía que trabajaba bien. Entonces, Beck decidió girar todas las perillas al máximo para ver que ocurría. Así fue como dio inicio a XP.

Los principios de XP citados verbatim de Beck:

• *El juego de Planeamiento*—Rápidamente determinar el alcance del próximo release mediante la combinación de prioridades del negocio y estimaciones técnicas. A medida que la realidad va cambiando el plan, actualizar el mismo.

• *Pequeños Releases*—Poner un sistema simple en producción rápidamente, luego liberar nuevas versiones en ciclos muy cortos.

• *Metáfora*—Guiar todo el desarrollo con una historia simple y compartida de cómo funciona todo el sistema.

• *Diseño Simple*—El sistema deberá ser diseñado tan simple como sea posible en cada momento. Complejidad extra es removida apenas es descubierta.

• *Testing*—Los programadores continuamente escriben pruebas unitarias, las cuales deben correr sin problemas para que el desarrollo continúe. Los clientes escriben pruebas demostrando que las funcionalidades están terminadas.

• *Refactoring*—Los programadores reestructuran el sistema sin cambiar su comportamiento para remover duplicación, mejorar la comunicación, simplificar, o añadir flexibilidad.

• *Programación de a Pares*—Todo el código de producción es escrito por dos programadores en una máquina.

• *Propiedad Colectiva del Código*—Cualquiera puede cambiar código en cualquier parte del sistema en cualquier momento.

• *Integración Continua*—Integrar y hacer builds del sistema varias veces por día, cada vez que una tarea se completa.

• *Semana de 40-horas*—Trabajar no más de 40 horas semanales como regla. Nunca trabajar horas extras durante dos semanas consecutivas.

• *Cliente en el lugar de Desarrollo*—Incluir un cliente real en el equipo, disponible de forma full-time para responder preguntas.

• *Estándares de Codificación*—Los programadores escriben todo el código de acuerdo con reglas que enfatizan la comunicación a través del mismo.

Como se observan, muchas de las prácticas propuestas contribuyen a maximizar la comunicación entre las personas, permitiendo de esa forma una mayor transferencia de conocimiento entre los desarrolladores y con el cliente, quien también es parte del equipo. Esto es logrado en la práctica gracias a la disposición física del lugar de trabajo. La idea es reunir a todas las personas en una misma oficina manteniendo una distribución denominada "cavernas y común" – ver Figura 002. En la misma se observan escritorios dispuestos en el centro con varias computadoras, cada una con dos sillas dispuestas de forma de permitir la programación de a pares. En las paredes se observan pequeños boxes o escritorios con sillas, los cuales pueden ser usados por los programadores en forma privada, para realizar llamados, consultar mail, o simplemente descansar la mente. Consecuentemente se logra el objetivo mencionado al inicio del párrafo de maximizar la comunicación y la transferencia de información en el área *común*, mientras que se mantiene la individualidad de las personas en las mencionadas *cavernas*.

Figura 002. Disposición física de las oficinas bajo la distribución "cavernas y común". Tomada de [Cockburn, 2001a]

Asimismo, XP impone un alto nivel de disciplina entre los programadores. El mismo permite mantener un mínimo nivel de documentación, lo cual a su vez se traduce

en una gran velocidad en el desarrollo. Sin embargo, una desventaja que deviene de esta falta de documentación es la incapacidad de persistir la arquitectura y demás cuestiones de análisis, diseño e implementación, aún después de que el proyecto haya concluido.

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

Si bien XP es la metodología ágil de más renombre en la actualidad, se diferencia de las demás metodologías que forman este grupo en un aspecto en particular: el alto nivel de disciplina de las personas que participan en el proyecto. XP será tratada en detalle más adelante en relación a algunas prácticas que comparte con AgEnD.

Scrum

Scrum define un proceso empírico, iterativo e incremental de desarrollo que intenta obtener ventajas respecto a los procesos definidos (cascada, espiral, prototipos, etc.) mediante la aceptación de la naturaleza caótica del desarrollo de software, y la utilización de prácticas tendientes a manejar la impredictibilidad y el riesgo a niveles aceptables. El mismo surge de un artículo de 1986 de la Harvard Business Review titulado “The New New Product Development Game” de Takeuchi y Nonaka, que introducía las mejores prácticas más utilizadas en 10 compañías japonesas altamente innovadoras. A partir de ahí y tomando referencias al juego de rugby, Ken Scwaber y Jeff Sutherland formalizan el proceso conocido como Scrum en el año 1995.

Uno de los análisis más importantes de la metodología desembocó en un libro escrito por dos de sus creadores, Ken Schwaber y Mike Beedle [Schwaber, 2001]. Este libro será tomado para el análisis de Scrum.

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.

Como se puede observar en la Figura 003 la metodología resulta sencilla definiendo algunos roles y artefactos que contribuyen a tener un proceso que maximiza el feedback para mitigar cualquier riesgo que pueda presentarse.

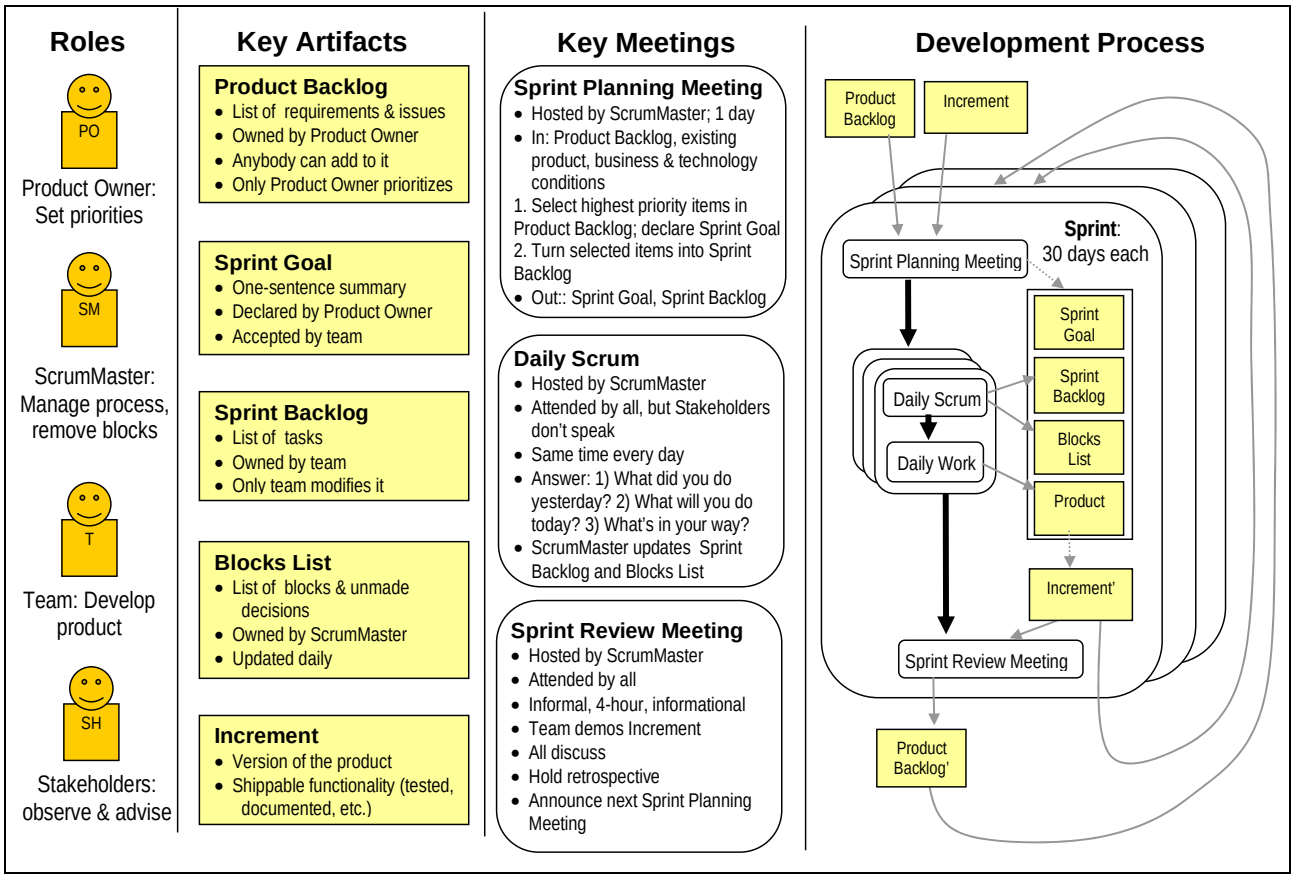

Figura 003. Descripción de roles, artefactos, reuniones y proceso de desarrollo de Scrum. Cortesía de **William C. Wake**, William.Wake@acm.org, www.xp123.com

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.

Crystal Clear

Alistair Cockburn es el propulsor detrás de la serie de metodologías Crystal. Las mismas presentan un enfoque ágil, con gran énfasis en la comunicación, y con cierta tolerancia que la hace ideal en los casos en que sea inaplicable la disciplina requerida por XP. Crystal "Clear" es la encarnación más ágil de la serie y de la que más documentación se dispone. La misma se define con mucho énfasis en la comunicación,

y de forma muy liviana en relación a los entregables [Cockburn, 2001b]. Crystal maneja iteraciones cortas con feedback frecuente por parte de los usuarios/clientes, minimizando de esta forma la necesidad de productos intermedios. Otra de las cuestiones planteadas es la necesidad de disponer de un usuario real aunque sea de forma part time para realizar validaciones sobre la Interfase del Usuario y para participar en la definición de los requerimientos funcionales y no funcionales del software.

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

Aunque al momento de preparar este trabajo aún no se dispone de bibliografía oficial de Crystal, Cockburn reporta su uso exitoso en diversos proyectos.

DSDM – Dynamic Systems Development Method

DSDM es la única de las metodologías aquí planteadas surgida de un Consorcio, formado originalmente por 17 miembros fundadores en Enero de 1994. El objetivo del Consorcio era producir una metodología de dominio público que fuera independiente de las herramientas y que pudiera ser utilizado en proyectos de tipo RAD (Rapid Application Development). El Consorcio, tomando las best practices que se conocían en la industria y la experiencia traída por sus fundadores, liberó la primera versión de DSDM a principios de 1995. A partir de ese momento el método fue bien acogido por la industria, que empezó a utilizarlo y a capacitar a su personal en las prácticas y valores de DSDM. Debido a este éxito, el Consorcio comisionó al Presidente del Comité Técnico, Jennifer Stapleton, la creación de un libro que explorara la realidad de implementar el método. Dicho libro [Stapleton, 1997] es tomado como guía para el análisis posterior de DSDM.

Dado el enfoque hacia proyectos de características RAD esta metodología encuadra perfectamente en el movimiento de metodologías ágiles. La estructura del método fue guiada por estos nueve principios:

1. El involucramiento del usuario es imperativo.
2. Los equipos de DSDM deben tener el poder de tomar decisiones.
3. El foco está puesto en la entrega frecuente de productos.
4. La conformidad con los propósitos del negocio es el criterio esencial para la aceptación de los entregables.
5. El desarrollo iterativo e incremental es necesario para converger hacia una correcta solución del negocio.
6. Todos los cambios durante el desarrollo son reversibles.
7. Los requerimientos están especificados a un alto nivel.
8. El testing es integrado a través del ciclo de vida.
9. Un enfoque colaborativo y cooperativo entre todos los interesados es esencial.

DSDM define cinco fases en la construcción de un sistema – ver Figura 004. Las mismas son: estudio de factibilidad, estudio del negocio, iteración del modelo funcional, iteración del diseño y construcción, implantación. El estudio de factibilidad es una pequeña fase que propone DSDM para determinar si la metodología se ajusta al proyecto en cuestión. Durante el estudio del negocio se involucra al cliente de forma temprana, para tratar de entender la operatoria que el sistema deberá automatizar. Este estudio sienta las bases para iniciar el desarrollo, definiendo las features de alto nivel que deberá contener el software. Posteriormente, se inician las iteraciones durante las cuales: se bajará a detalle los features identificados anteriormente, se realizará el diseño de los mismos, se construirán los componentes de software, y se implantará el sistema en producción previa aceptación del cliente.

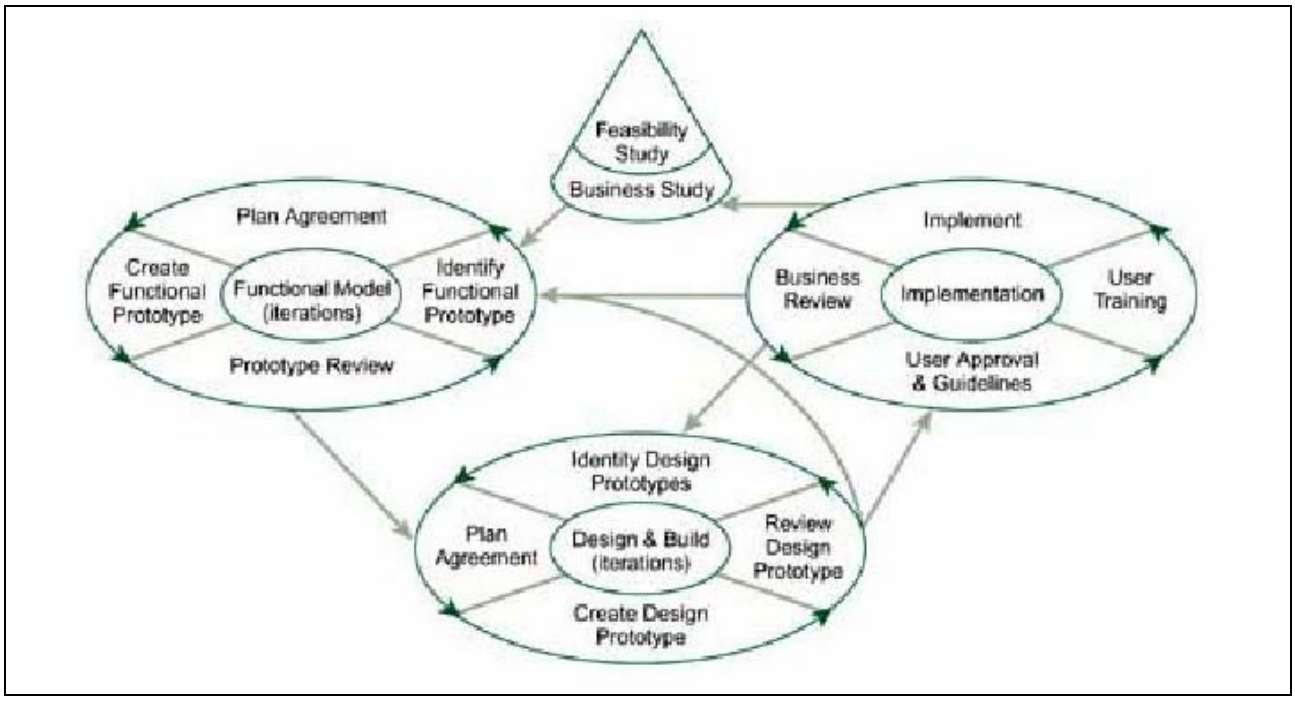

Figura 004. Fases del proceso de desarrollo DSDM. Tomada de [Highsmith, 2002].

Descontando la primera fase que es realizada una única vez al principio del proyecto para analizar la factibilidad desde el punto de vista del negocio del desarrollo, las demás fases presentan las características del modelo iterativo e incremental ya tratado. Sin embargo, lo que diferencia a DSDM de dicho modelo son los principios alrededor de los cuales se estructura y que hacen énfasis en los equipos de desarrollo, en el feedback con el cliente, en las entregas frecuentes de productos.

Para resolver la cuestión de la aplicabilidad de DSDM a un proyecto convendrá responder las siguientes preguntas:

¿Será la funcionalidad razonablemente visible en la interfase del usuario?

¿Se pueden identificar todas las clases de usuarios finales?

¿Es la aplicación computacionalmente compleja?

¿Es la aplicación potencialmente grande? Si lo es, ¿puede ser particionada en componentes funcionales más pequeños?

¿Está el proyecto realmente acotado en el tiempo?

¿Son los requerimientos flexibles y sólo especificados a un alto nivel?

Las mismas refieren a las características que se deben cumplir en los proyectos para poder utilizar el enfoque RAD de construcción. Se observa que aquellos proyectos que califiquen afirmativamente de acuerdo a dichas preguntas tendrán las siguientes características que refieren a la aplicabilidad de DSDM:

- Son proyectos interactivos con la funcionalidad visible en la interfase de usuario
- De baja o media complejidad computacional
- Particionables en componentes de funcionalidad más pequeños si la aplicación es de gran tamaño
- Acotados en el tiempo
- Con flexibilidad en los requerimientos
- Con un grupo de usuarios bien definidos y comprometidos al proyecto

De esta forma observamos que DSDM deja las bases sentadas para el análisis sobre su aplicabilidad a un espectro bien definido de proyectos de software. Sin embargo, la metodología no tiene ninguna prescripción respecto a las técnicas a ser usadas en el proyecto, ni siquiera impone el desarrollo bajo un paradigma específico – funciona tanto para el modelo de orientación a objetos como para el modelo estructurado. Algo que sí sugiere el método es la generación de un conjunto mínimo de modelos necesarios para la sana progresión de la entrega del software y facilidad en el mantenimiento. Estos modelos esenciales deberán ser definidos antes que comience el desarrollo, y deberán ser revisados en las sucesivas iteraciones para validad su contenido.

El concepto de timebox es algo que está embebido en DSDM y en todas las metodologías ágiles, en las cuales también se conocen como iteración, ciclo, intervalo. La consecuencia de utilizarlos es el feedback frecuente que brinda visibilidad a los stakeholders para que verifiquen el progreso y puedan tomar acciones correctivas a tiempo. También permiten controlar la calidad de los productos intermedios que se van generando, y realizar estimaciones de esfuerzo más precisas. Asimismo, cada timebox esta compuesta por actividades definidas en relación a entregables en vez de tareas.

Cada entregable generado durante el mismo es testeado/revisado dentro del mismo timebox.

En DSDM, un timebox consta de tres fases que son: *Investigación*, *Refinamiento* y *Consolidación*. Durante la *Investigación* se chequean que las actividades que componen el timebox se condicen con la arquitectura del sistema. Esta es una fase de carácter exploratorio, en la que se fijan los objetivos de la iteración, los entregables a ser producidos, efectuándose revisiones sobre las iteraciones anteriores a la actual. La siguiente fase, *Refinamiento*, consiste en la producción propiamente dicha de los artefactos planificados. DSDM destaca la necesidad de colocar componentes de distinta prioridad en un mismo timebox, de manera de poder posponer a futuras iteraciones aquellos con menor prioridad, en caso que surjan imprevistos o se materialicen riesgos. Finalmente, la fase de *Consolidación* consiste en completar los entregables, verificando la calidad de los mismos. En esta fase que posee el hito de finalización del timebox se demostrará que se satisficieron los requerimientos de calidad definidos durante la *Investigación*.

DSDM incluye roles claves en relación al management del proyecto. Identifica al *visionario* como el encargado de asegurar que se satisfacen las necesidades del negocio; el *usuario embajador* que equivaldría al *on-site customer* de XP, que brinda el conocimiento del negocio y define los requerimientos del software; el *coordinador técnico* que es la persona encargada de mantener la arquitectura y verificar la consistencia de los componentes construidos respecto a esta y al cumplimiento de los estándares técnicos.

Algunas técnicas sugeridas en DSDM son las sesiones JAD para capturar los requerimientos del software y la realización de prototipos para descifrar aquellas ambigüedades que se presentan en el relevamiento y también para derribar las barreras comunicacionales entre analistas y usuarios. El enfoque propuesto consiste en la utilización de un prototipo evolutivo, el cual se va refinando hasta tenerse la aplicación deseada. El énfasis queda en manifiesto en los prototipos que se sugieren para cada etapa: negocio, usabilidad, performance y capacidad, y diseño.

En resumen, encontramos en DSDM una metodología ágil creada en el Reino Unido a partir de un consorcio con participación de empresas de primera línea. El mismo contiene las características principales de las metodologías ágiles y contiene

prácticas tendientes al enfoque RAD. Algo que es importante de DSDM ha sido su aceptación en la industria y su refinamiento continuo – actualmente, se encuentra en su versión 4.0 – lo que indica que las metodologías ágiles no son solo dominio de pequeños grupos de desarrollo sino que están siendo adoptadas por "pesos pesados" en las industrias.

FDD – Feature Driven Development

Peter Coad es considerado uno de los referentes más importantes dentro de la ingeniería de software. Coad ha sido uno de los principales pioneros detrás del movimiento de la orientación a objetos y empezó a trabajar con Ed Yourdon (uno de los creadores del Análisis Estructurado) a principios de los noventa, cuando este último pidió ayuda a alguien de la comunidad de objetos para desarrollar una nueva metodología, basada en el paradigma de OO. Posteriormente, Coad junto con Jeff De Luca y otros, participaría en la creación de FDD, una metodología desarrollada alrededor del año 1998 que presenta las características de un proceso ágil [Coad, 1998]. La misma derivó del trabajo de Coad sobre las *Feature Lists (Listas de Funcionalidades).*

FDD se estructura alrededor de la definición de features que representan la funcionalidad que debe contener el sistema, y tienen un alcance lo suficientemente corto como para ser implementadas en un par de semanas. FDD posee también una jerarquía de *features*, siendo el eslabón superior el de *feature set* que agrupa un conjunto de *features* relacionadas con aspectos en común del negocio. Por último, establece el *major feature set* como el más alto nivel de agrupación de funcionalidad que abarca diferentes *feature sets* que contribuyen a proveer valor al cliente en relación a un subdominio dentro del dominio completo de la aplicación. Una de las ventajas de centrarse en las *features* del software es el poder formar un vocabulario común que fomente que los desarrolladores tengan un diálogo fluido con los clientes, desarrollando entre ambos un modelo común del negocio. Este tema será tratado más adelante en relación al enfoque de las metodologías ágiles en los productos entregados.

La jerarquía de los *features* utiliza los siguientes formatos:

- Para features: <**acción**> el <**resultado**> <**de | para | sobre | por**> un <**objeto**>
- Para feature sets: <**acción**><**-endo**> un <**objeto**>
- Para major feature sets: administración de <**acción**>

Ejemplos:

- Calcular el total de la facturación de Noviembre (feature)
- Modificar el estado de las facturas de producción (feature)
- Administrar el perfil de los proveedores (feature)
- Haciendo una venta a un cliente (feature set)
- Cargando la facturación de los proveedores (feature set)
- Administración de Bancos (mayor feature set)

El ciclo de vida propuesto por FDD se puede observar en la Figura 005 y está compuesto por cinco procesos, dos de las cuales se realizan tantas veces como iteraciones se planifiquen en el desarrollo.

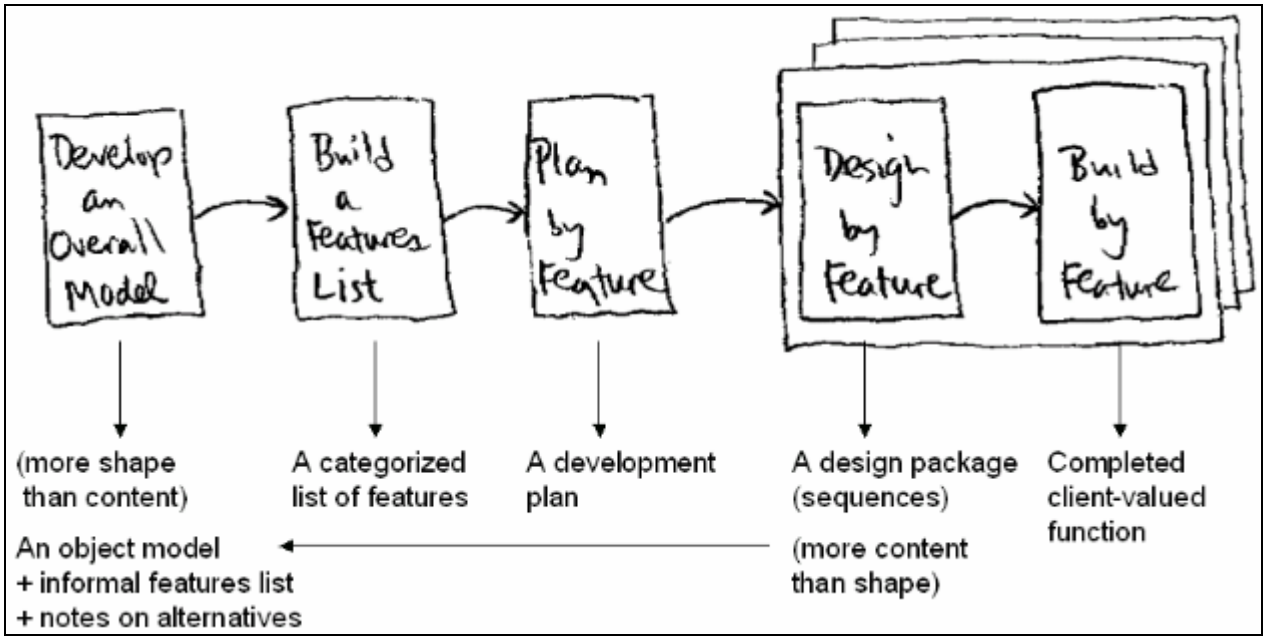

Figura 005. Los cinco procesos dentro de FDD. Tomada de [Coad, 2000].

La primera actividad consiste en *Desarrollar un Modelo Global*, que sugiere un cierto paralelismo con la construcción de la arquitectura del software. En la creación de este modelo participan tanto los expertos en el dominio como los desarrolladores. Mediante el esfuerzo de ambas partes se intenta lograr lo que el modelo en espiral proponía con sus primeras iteraciones: un conocimiento global de la aplicación a construir, el entendimiento del negocio en que esta embebida, un primer bosquejo de las *features* del software, y la definición de restricciones y cuestiones no funcionales. Para esto, se desarrollarán: diagramas de los paquetes, con las clases esenciales y las responsabilidades de las mismas; un documento similar al de Visión en donde se plasmen los objetivos del proyecto y como el mismo ayuda al negocio; un documento con los requerimientos no funcionales detectados; por último, el documento que podríamos llamar *arquitectura* y en el que figuran las opciones de modelado surgidas durante esta actividad.

La segunda actividad, *Construir una Lista de Features*, comienza tomando el bosquejo de *features* formulado durante la actividad anterior para refinar las funcionalidades incluidas. Una vez que se han identificado las mismas se las agrupa jerárquicamente para poder estructurar el trabajo de desarrollo; se realiza la priorización de las mismas basándose en la satisfacción al cliente – las prioridades sugeridas para las *features* por FDD son: A (debe tener), B (sería útil tener), C (agregar si es posible), o D (futuro); finalmente, se pondera la importancia de cada una para su posterior implementación – en caso que existan *features* que requieran más de dos semanas de desarrollo en esta actividad se particionarán para lograr ubicarlos en iteraciones.

La tercera actividad, *Planificar por Feature*, toma como input la lista priorizada de la fase anterior y establece los tiempos las futuras iteraciones. En esta actividad participan el líder de proyecto, el líder de desarrollo y el programador jefe. A medida que se realiza la planificación se delinean los hitos de finalización de las iteraciones, dejando asentado cuales son los *features* y *features sets* que estarán construidos en dichos hitos. Parte de también incluye la delegación de responsabilidades a los programadores jefe que serán dueños de los *features*, estos a su vez asignarán las clases a dueños de clases seleccionados del equipo.

Las últimas dos actividades, *Diseñar por Feature* y *Construir por Feature*, están relacionadas con la parte productiva del proceso en que se construye la aplicación de manera incremental. Empezando por el diseño que toma los *features* correspondientes a

la iteración, el equipo de programadores liderado por el programador jefe identifica las clases, atributos y métodos que realizan la funcionalidad requerida. Mediante la utilización de diagramas de secuencia de UML, se verifica que el diseño pueda ser implementado. Se realizará también una inspección del diseño en los casos en que la complejidad de la funcionalidad lo requiera. Posteriormente, en la fase de *Construir por Feature*, se procede a desarrollar las clases definidas en la actividad anterior. Cada programador implementará los métodos de las clases por las que este es responsable, extendiendo las clases base de prueba para construir las pruebas unitarias. Una vez que la clase pasa todas las pruebas, se inspecciona el código. Esta actividad será realizada por el equipo asignado al *feature* en cuestión, y una vez que finaliza se promueve el código al Build correspondiente, siendo entregado a Administración de la Configuración.

En relación a las actividades de management en FDD se recomienda una reunión semanal entre el Líder de proyecto y los programadores jefe; la misma debe ser breve, de no más de 30 minutos, y en la cual se reporta el status de los *features* que están siendo construidos por cada grupo. Por cada *feature set* que es implementado se tendrá la información como indica la Figura 006 para medir el avance del proyecto, dándole visibilidad al management superior y al cliente.

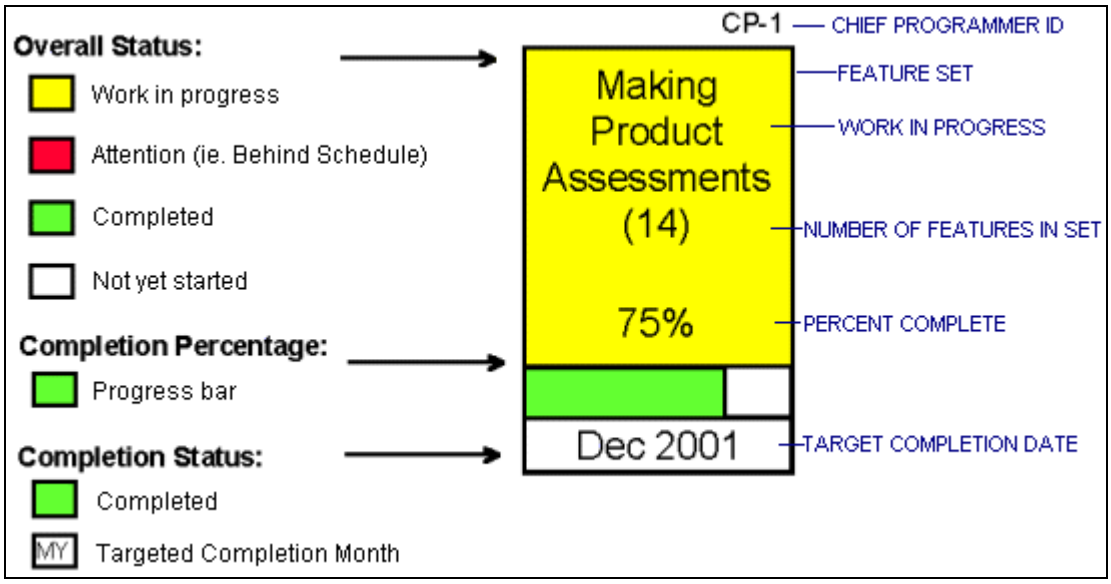

Figura 006. Reportando el progreso al management y al cliente en FDD. Tomada de [Coad, 2000].

En conclusión, encontramos en FDD un proceso ágil orientado a la funcionalidad del software por sobre las tareas, sobre las cuales da guías mínimas. El

proceso sugiere organizar bloques de *features* a ser construidos en forma incremental mediante iteraciones de dos semanas; provee estrategias de planeamiento para el líder de proyecto; fomenta la colaboración mediante la creación de equipos dirigidos por un programador jefe.

ASD – Adaptive Software Development

Jim Highsmith en su libro [Highsmith, 1999] es la mente detrás de este proceso ágil. ASD consiste en un cambio de filosofía en las organizaciones pasando de la transición del modelo Comando-Control al modelo Liderazgo-Colaboración. Basado en los conceptos de los *Sistemas Adaptativos Complejos* relacionada con la Inteligencia Artificial, Highsmith lleva los mismos al campo de la Ingeniería de Software en particular. Dada la complejidad inherente al software concluye que la aplicación de esta teoría es esencial para el nuevo escenario que plantea la economía global.

Comenzando por un cambio en el modelo de desarrollo determinista, tomado del ciclo de Deming, en que se aplica la secuencia Planificar-Ejecutar-Evaluar. Dicho esquema es llevado a la práctica con el modelo en cascada, en que se realiza una precisa planificación inicial mediante el WBS, el Gantt, y el Pert definiendo las tareas a realizar en detalle, luego se tiene las fases de construcción, y finalmente, se tiene el testing que brinda el feedback en relación al producto construido.

ASD propone utilizar en cambio el ciclo de vida de la Figura 007, Especular-Colaborar-Aprender. El proyecto comienza con una fase de especulación en que en que se lleva a cabo la planificación tentativa del proyecto en función de las entregas que se irán realizando. La utilización del verbo Especular demuestra el interés de Highsmith en demostrar la naturaleza impredecible de los sistemas complejos. En esta etapa se fija un rumbo determinado a ser seguido en el desarrollo, sabiendo a partir de ese momento que no será el lugar en que finalizará el proyecto. En cada iteración, se aprenderán nuevas funcionalidades, se entenderán viejas cuestiones, y cambiarán los requerimientos. Gracias a centrarse en la especulación, ASD permite administrar estos proyectos de alto cambio y rápido desarrollo que se encuentran en el *borde del caos*. Respecto a la especulación, se recomienda realizar un *component breakdown structure* en vez del muy conocido y tradicional *work breakdown structure (WBS)* en el cual mediante una grilla u hoja de cálculo se pueda conocer la funcionalidad a ser liberada en cada ciclo.

Sin embargo, no es más que una especulación ya que el carácter adaptativo del proceso permite pequeñas desviaciones en un sentido – por lo que Highsmith sugiere que cada ciclo se componga de un mix entre funcionalidades críticas, útiles, y opcionales, previendo los posibles retrasos que puedan existir mediante el movimiento de las funcionalidades de menor prioridad a futuros ciclos – y grandes desviaciones en otro, las cuales son utilizadas para la exploración del dominio y de la aplicación, que puede llevar a cambiar el rumbo del proyecto – estos desvíos está representado por las flechas de divergencia en la Figura 007.

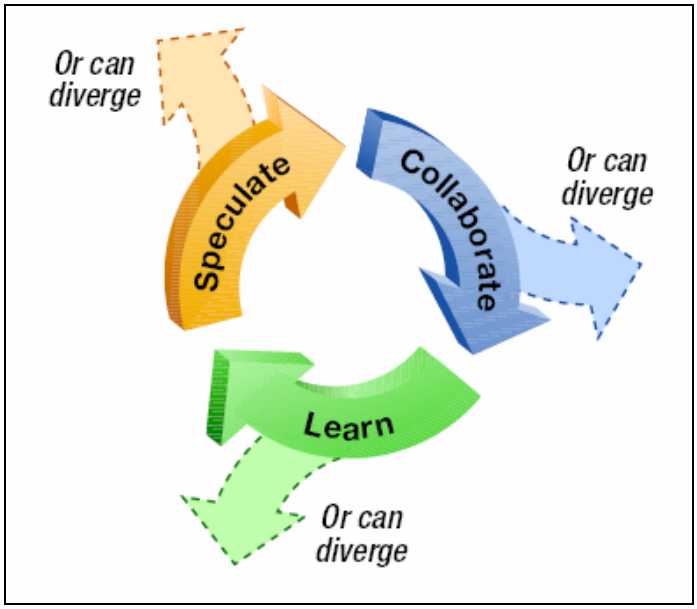

Figura 007. El ciclo de vida adaptativo. Tomada de [Highsmith, 2000].

La siguiente fase del ciclo de vida, Colaborar, es aquella en la que se construye la funcionalidad definida durante la especulación. ASD define un *Componente* como un grupo de funcionalidades o entregables a ser desarrollados durante un ciclo iterativo. Durante cada iteración el equipo colabora intensamente para liberar la funcionalidad planificada. También, existe la posibilidad de explorar nuevas alternativas, realizar pruebas de concepto, pudiendo eventualmente alterar el rumbo del proyecto profundamente. ASD no propone técnicas ni prescribe tareas al momento de llevar a cabo la construcción simplemente mencionando que todas las prácticas que sirvan para reforzar la colaboración serán preferidas, siguiendo de esta forma la línea de las metodologías ágiles respecto a la orientación a componentes. El énfasis se ubica en la relaciones entre las personas que deben estar lo suficientemente lubricadas para generar una propiedad imprescindible de los organismos complejos: emergencia. La emergencia

es una propiedad de los sistemas adaptativos complejos que crea alguna propiedad más grande del todo (comportamiento del sistema) a partir de la interacción entre las partes (comportamiento auto-organizativo de los agentes). Gracias a esta propiedad los grupos de desarrollo logran sacar lo mejor de si en la el *borde del caos*.

La fase final de ASD, Aprender, consiste en la revisión de calidad que se realiza al final de cada ciclo. En la misma se analizan cuatro categorías de cosas para aprender [Highsmith, 2000]:

- Calidad del resultado de la desde la perspectiva del cliente
- Calidad del resultado de la desde la perspectiva técnica
- El funcionamiento del equipo de desarrollo y las prácticas que este utiliza
- El status del proyecto

Para evaluar la calidad desde el punto de vista del cliente se sugieren utilizar grupos de enfoque en el cliente, mediante los cuales se explora un modelo de la aplicación y se anotan los requerimientos de cambio del cliente.

Las revisiones al diseño, al código o a las pruebas permitirán aprender sobre la calidad de los mismos. En este caso, el énfasis estará puesto en aprender cuales han sido los errores o desvíos y poder resolverlos, y no en encontrar culpables. Asimismo, está es la etapa en que se evaluarán las exploraciones que se hayan realizado dando la capacidad de poder modificar la arquitectura del sistema si se ha encontrado algún camino que se ajusta mejor a lo que necesita el usuario o si han cambiado los requerimientos.

El tercer proceso de feedback está relacionado con la interacción entre las partes, la dinámica de grupo, y las técnicas empleadas. Para medir la performance y el grado de cohesión del mismo, se podrán realizar al final de cada ciclo pequeñas reuniones de postmortem. En las mismas se discuten los aspectos del proceso que contribuyen al desarrollo y aquellos que deben ser descartados por su influencia negativa.

En relación al status del proyecto, se realizarán revisiones para determinar el estado del mismo en relación a lo planificado. En este momento, se detectarán posibles diferencias que pueden surgir de la exploración y que cambiarán el rumbo a que apuntaba el proyecto.

En la Figura 008 se puede ver el detalle interno de cada fase como ya fue explicado, mostrándose con una flecha que trasciende las tres fases en sentido inverso, el bucle de aprendizaje. Este bucle es algo crítico para ASD ya que denota un cambio en el esquema tradicional de la vista de un sistema en que se tenía un bucle de control para detectar diferencias y corregirlas. Es decir, en las metodologías tradicionales las diferencias respecto a lo planificado eran vistas como errores que debían ser enmendados para que cumplieran lo pautado. ASD y las metodologías ágiles plantean la necesidad de que el feedback necesario sea para aprender, nos da la posibilidad de entender más respecto al dominio y construir la aplicación que mejor satisfaga las necesidades del cliente. Highsmith lo expone claramente en la siguiente frase:

> **En ambientes complejos, el seguir un plan al pie de la letra produce el producto que pretendíamos, pero no el producto que necesitamos.**

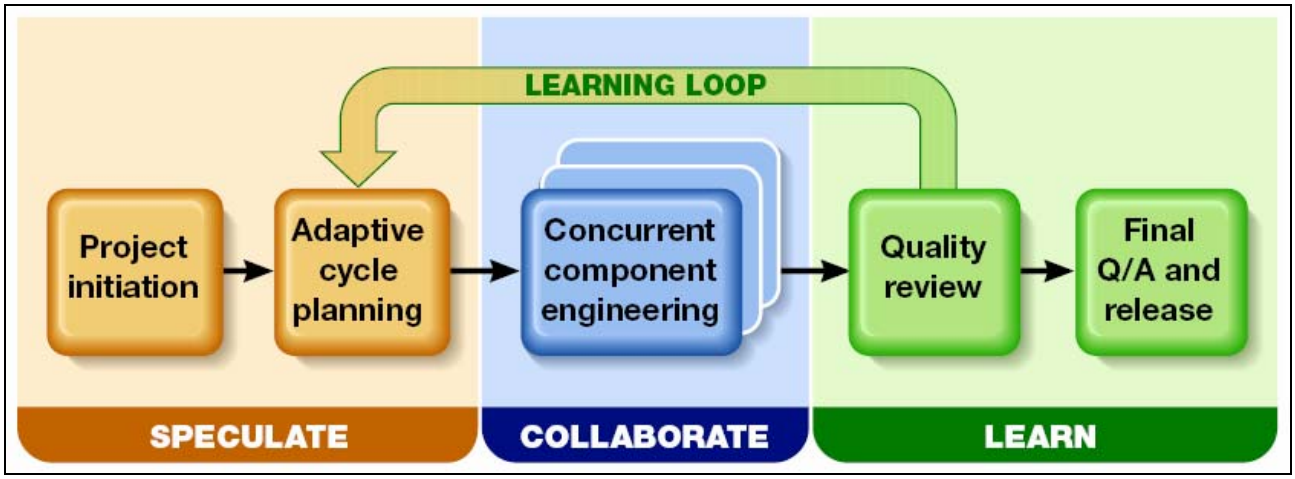

Figura 008. Actividades del ciclo de vida adaptativo. Tomada de [Highsmith, 2000].

En conclusión, tenemos en ASD un marco filosófico basado en la teoría de *Sistemas Adaptativos Complejos* que nos permite encarar la construcción de software en forma ágil utilizando las prácticas que nos resulten convenientes en cada caso. En este sentido resulta similar a Scrum.

XBreed – Scrum wrapper for XP

XBreed ofrece una de las primeras fusiones exitosas entre metodologías ágiles. Tomando las fuerzas en el management de Scrum y uniéndolas con las prácticas de XP nace una de las más modernas metodologías alrededor del 2000. Mike Beedle es la mente detrás de XBreed y su objetivo es tomar las mejores prácticas del universo ágil y unificar estas en un marco de trabajo que permita el desarrollo concurrente de múltiples proyectos usando metodologías ágiles.

Al momento de escritura de esta tesis, existe muy poca información de XBreed, salvo el sitio web oficial del mismo.

Manifesto for Agile Software Development

El Manifiesto para el Desarrollo Ágil de Software es de suma importancia dentro del movimiento de las metodologías ágiles. El mismo representa una iniciativa conjunta entre los principales responsables de los procesos ágiles mencionados anteriormente para lograr unificar principios compartidos por las diversas metodologías de manera de crear un framework de trabajo que contribuya al mejoramiento del desarrollo ágil.

Uno de los principales objetivos del encuentro en que se generó el Manifiesto fue el de extraer un factor común de los principios esenciales que servirían de guía para cualquier metodología que se identifique como ágil. Esto concluyó en la declaración de lo que podríamos denominar el prólogo del Manifiesto [Manifesto, 2001]:

> **Estamos descubriendo mejores maneras de desarrollar software mediante su construcción y ayudando a que otras personas lo construyan.**
>
> **A través de este trabajo hemos llegado a valorar:**
>
> **Individuos e interacciones sobre procesos y personas**
>
> **Software funcionando sobre documentación comprensiva**
>
> **Colaboración del cliente sobre negociación de contrato**
>
> **Responder al cambio sobre seguir un plan**

Esto es, mientras que existe valor en los ítems de la derecha, valoramos más los ítems de la izquierda.

Analizando los factores mencionados en detalle vemos que los mismos forman parte esencial de las metodologías descriptas. En los mismos se plasman aquellas razones que impulsaron a Beck, Highsmith, y otros, a construir metodologías ágiles. El énfasis sobre las personas esta emblematizado en la primer oración respecto a los valores de las metodologías. El énfasis en el código por sobre los documentos es una propuesta eficaz para lograr el rápido feedback requerido por la agilidad. La participación del cliente es fundamental para el éxito de los proyectos ya que ellos definen la funcionalidad a construir y guían el desarrollo de las pruebas funcionales. La importancia de responder al cambio indica que el desarrollo de software no es más que un proceso de aprendizaje en que cada cambio nos permite conocer más en detalle el dominio de la aplicación.

El autor entiende que para que se pueda definir a AgEnD como una metodología ágil, la misma deberá estar completamente circunscripta a los valores detallados en el Manifiesto. Es por esto que se harán continuas referencias al mismo para comentar la conformidad del proceso a los principios y valores.

Con esto, damos por concluida la introducción de este documento. A partir de ahora, nos centraremos en la descripción de AgEnD, su aplicabilidad en el desarrollo de software, sus valores, principios, y toda la información necesaria para poder utilizarla en la práctica; posteriormente, plantearemos un caso de estudio de AgEnD en donde se pueda observar y analizar sus fuerzas y sus debilidades; finalmente, se llevaran a cabo aquellas conclusiones relevantes al desarrollo de la AgEnD, analizándose las limitaciones y alcances de la misma, previendo posibles líneas de trabajo que se puedan desprender en relación a las metodologías ágiles y el Manifiesto que las nuclea.

Presentación de AgEnD

Como se ha mencionado en el Abstract, el propósito de esta tesis es construir una metodología que reúna aquellas mejores prácticas del universo ágil uniéndolas con un framework que ayude a las organizaciones a implantar este tipo de metodologías.

El nombre de la metodología es *AgEnD* que significa en inglés *Agility Enhanced Development* (Desarrollo Mejorado Por Agilidad). En otras palabras, la idea es configurar un proceso de desarrollo que resulte fácil de implementar y basar el mismo en la aplicación de prácticas ágiles para contribuir a la construcción del software, lidiando con la impredictibilidad de la disciplina.

En esta tesis se comentará el porqué de la necesidad de un proceso, y de cómo AgEnD llena ese hueco que surge al querer implementar una metodología ágil en una organización.

Capítulo II - Descripción del Problema

A continuación se describirán las razones por las cuales hoy en día se debe contar con un proceso de desarrollo. También se explicará como las metodologías ágiles intentan resolver distintos problemas que se plantean en el desarrollo de software.

Por qué elegir un proceso

Cuando una empresa encara proyectos de desarrollo de software, ¿que la impulsa a seleccionar un proceso? ¿No alcanza con dejar que el conocimiento y el esfuerzo de los involucrados sea aplicado en forma uniforme, y simplemente que se junten los frutos del trabajo de las personas al final del proyecto?

Estas simples preguntas no poseen una única respuesta, pero sí se puede argumentar la siguiente afirmación que indica las posibles opciones en relación a la elección de un proceso.

> **La elección respecto a la utilización o no de un proceso depende principalmente del grado de predictibilidad que se desee tener en el desarrollo.**

Es decir, si una empresa está interesada en que cada proyecto sea como una aventura, en que se desconozca el resultado y existan una gran cantidad de riesgos, no necesitará proceso alguno. Simplemente dejará en manos de los profesionales de sistemas el desarrollo de proyectos mediante las técnicas que utilice cada individuo en su trabajo. La aleatoriedad de cada proyecto será una característica inherente. Se realizarán planificaciones a muy corto plazo, ya que no existe visibilidad para las personas externas al equipo de desarrollo. En conclusión, estamos hablando de todas las organizaciones que se encuentran en el nivel 1 del CMM, el denominado nivel *Inicial*. Indudablemente, cualquier organización que se precie de ser tal jamás elegiría este enfoque como primera instancia ya que uno de los objetivos de la misma sería *maximizar ganancias a largo plazo* y para esto debería mantener procesos que aseguren el éxito en todos los sectores en que está compuesta. Sin embargo, como veremos más adelante la mayoría de las empresas termina eligiendo esta opción sin ser plenamente conciente de la elección y de los resultados que esta acarrea.

Por otro lado, encontramos organizaciones que poseen manuales de procedimientos para cada una de las tareas relacionadas con el desarrollo de software. Cada actividad realizada por cada persona está descripta en detalle en forma escrita y en cualquier momento esta documentación podrá ser consultada si tienen dudas respecto a algún punto del proceso. En general, estas organizaciones poseen una estructura

tradicional de management en donde el modelo Comando-Control es utilizado en todos los niveles jerárquicos. De características deterministas, relacionadas muchas veces con la manufactura de productos de alguna clase, conducen el departamento de sistemas de la misma forma en que conducen el departamento de finanzas, utilizando rígidos procesos que generan hitos y entregas bien definidos. Bajo esta perspectiva es lógico concluir que una organización que posee procedimientos predecibles en todas sus áreas, intente llevar los mismos al desarrollo de software. Sin embargo, como hemos notado en la introducción esto resulta una tarea imposible dadas las características esenciales del software mencionadas por Brooks.

Analizando estos dos enfoques de desarrollo de software nos encontramos con dos extremos bien definidos por [Highsmith, 1999] que son el de *Burocracia*, representado por las organizaciones con procesos rígidos y definidos hasta el más mínimo nivel de detalles, y el de *Adhocracia*, que representa el desarrollo caótico sin ningún proceso ni visibilidad sobre el estado y el rumbo de los proyectos. En un extremo, estamos en la claridad cegante de un proceso tan pautado que termina sin satisfacer las necesidades de los usuarios, y en el otro estamos en la oscuridad completa y aleatoriedad total que deviene del caos permitiendo únicamente beneficios o pérdidas a corto plazo.

Si trazáramos una línea uniendo la *Burocracia* con la *Adhocracia* podríamos utilizarla para ubicar el grado de formalidad dado por las metodologías que ya fueron tratadas en la introducción. Es decir, si tenemos una organización que se identifique con uno de estos enfoques, cuáles serían las metodologías que elegiría para el desarrollo de software. En la Figura 009 observamos la disposición que surge de colocar al Modelo en Cascada, el RUP, las Metodologías Ágiles, el Modelo en Espiral, el Modelo por Prototipos, y el Modelo Iterativo en la recta correspondiente. En la misma observamos que el Modelo en Cascada es el que más contribuye a un régimen burocrático en el que los proyectos son planificados en base a tareas a realizar por los recursos; fomenta el orden de la organización mediante la clara división en etapas con sus inputs/outputs correspondientes. Asimismo, encontramos otro de los procesos más utilizados en la actualidad como es el RUP, en una forma completa, construyendo todos los artefactos especificados y realizando todas las actividades planteadas. En otras palabras, estos procesos pretenden definir todas las actividades, roles, entregables, y demás aspectos relacionados en un proyecto de software, para generar resultados predecibles a lo largo

del tiempo. Mediante la clara definición de aquellas cuestiones involucradas y la posibilidad de controlar el proceso para ajustarlo ante posibles desvíos, se llega a la *Burocracia*. El Modelo en cascada ejemplifica este enfoque al ser un modelo orientado a documentos, en donde el progreso se mide en base a un conjunto de fases bien definidas y al grado de avance de la documentación correspondiente. Las personas son partes reemplazables en este esquema, y como tales alcanza con definir los roles y las actividades que estas realizan para completar el marco de desarrollo.

A medida nos alejamos de la *Burocracia* encontramos los Modelos con un enfoque principal hacia los entregables y hacia el rápido feedback de los clientes. El RUP Agilizado no es otra cosa que una versión del RUP customizada para grupos pequeños y con feedback frecuente de los interesados.

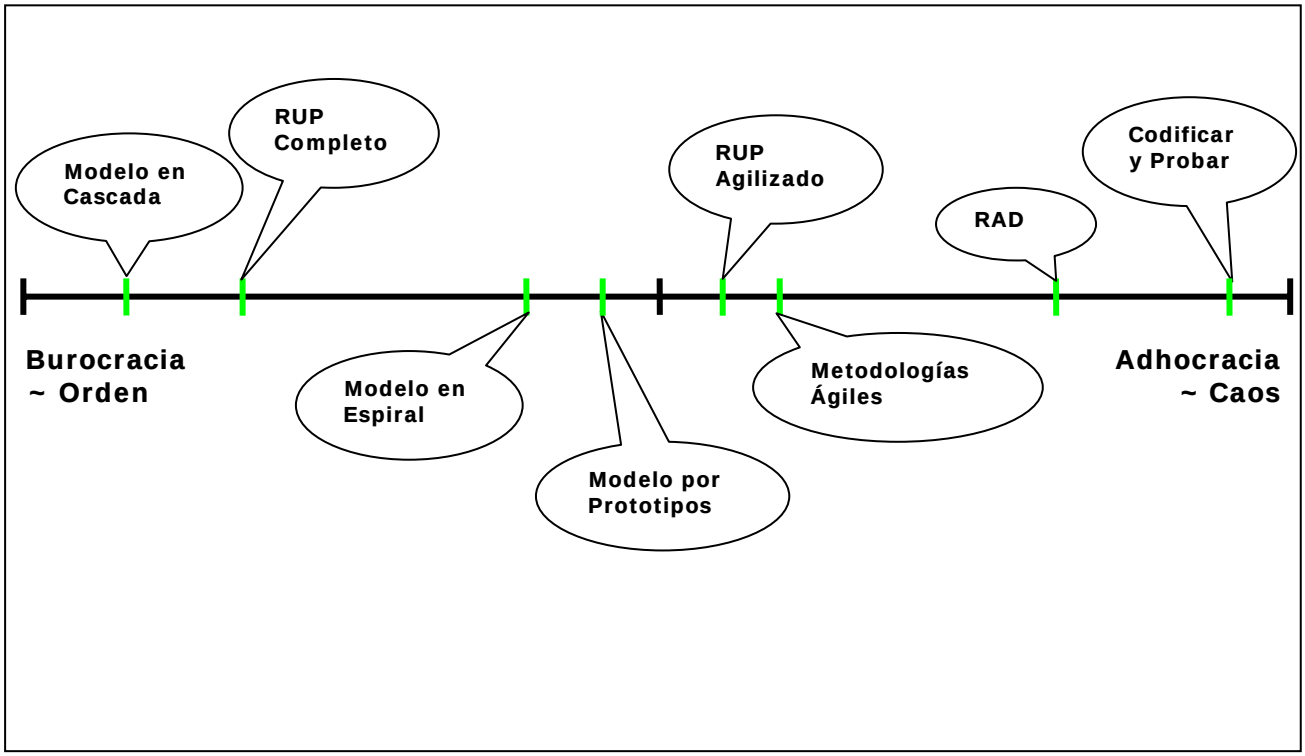

Figura 009. Ubicación de las metodologías en relación al nivel de formalidad requerido.

Como observamos, todas estas metodologías se encuentran hacia la izquierda del punto medio entre *Burocracia* y *Adhocracia*. Si nos movemos desde ese punto hacia el Caos nos encontramos con el tema central de esta tesis, las Metodologías Ágiles, entre las que encontramos todas aquellas descriptas en la Introducción, incluidas AgEnD. Estas metodologías reconocen las características inherentes de complejidad del software, y el carácter empírico que debe tener un proceso de desarrollo del mismo. En

estas, las personas son de primera magnitud y se prefiere sacrificar el proceso en pos de darles libertad a las personas. Sin embargo, esto no significa que se termina en un proceso de Codificar y Probar, liderado por programadores rebeldes que se pasan el día generando código. Es por esto que el punto en que se ubican las mismas está bastante alejado del Caos en que la aleatoriedad es la norma. Otra cuestión que diferencia estos procesos de los anteriores es el cambio existente entre un proceso empírico y un proceso determinista o definido. Como mencionábamos antes, los procesos más burocráticos tienden a tener definidos todos los aspectos del desarrollo. Esto se contradice con la naturaleza compleja del software, para lo cual se adaptaría con mayor eficiencia un proceso empírico. Por esto nos referimos a un proceso en que se desconocen muchos aspectos, se reconocen a las personas como componentes de primer orden totalmente impredecibles, y simplemente se intenta guiar el desarrollo hacia un objetivo que puede no permanecer constante en el tiempo a medida que aumenta el conocimiento de la aplicación a ser construida. En los procesos empíricos se conocen las buenas prácticas probadas con la experiencia, y los resultados u outputs que surgen generalmente de tomar ciertos inputs o de realizar ciertas actividades. Estos modelos surgen de la observación y la experiencia obtenida a lo largo de los años, sin basarse en leyes de la naturaleza o principios teóricos fundamentados. Es por esto que se los asocia a un enfoque pragmático para desarrollar software.

Siguiendo por la recta nos encontramos con el enfoque RAD propuesto por [Martin, 1991] en el que se tenía un lenguaje de alto nivel lo suficientemente poderoso como para permitir generar código de forma rápida, un pequeño equipo trabajando en conjunto con buenos canales de comunicación, y un plan de entregas de funcionalidad en forma iterativa. El mismo no poseía pautas claras en relación a la calidad del producto, la satisfacción de los requerimientos de los usuarios, la clara definición de la arquitectura del sistema. Por estas causas terminó siendo asociado con la generación rápida de aplicaciones con bajos niveles de calidad, y que resultaban en pesadillas de mantenimiento. Sin embargo, debemos reconocer el aporte que hicieron a la ingeniería de software ya que gracias a estas resurgieron las herramientas CASE y comenzó el uso masivo de lenguajes de alto nivel, sirviendo como base para el posterior advenimiento de las metodologías ágiles.

Finalmente, nos encontramos justo en las cercanías del Caos, al modelo Codificar y Probar. Esto modelo, caótico por excelencia, se basa en el siguiente flujo de

actividades: un programador trabajando en su computadora, genera código fuente; una vez que termina, lo compila, corrige los errores y lo vuelve a compilar; cuando ya no tiene más errores, genera el ejecutable y lo corre; ahí prueba el correcto funcionamiento del mismo, debuggeando el código al instante hasta corregir aquellos errores que detecta; cuando este proceso iterativo finaliza, continúa con el siguiente componente/módulo/programa. El mismo es totalmente impredecible y la visibilidad para las personas involucradas en el mismo es inexistente. Es este modelo el que se trata de evitar a toda costa al formular una metodología, incluidas las metodologías ágiles.

Corolarios

Del anterior análisis deberemos tener en cuenta ciertos aspectos inherentes a los proyectos de desarrollo, que incidirán en forma directa en el grado de predictibilidad de la metodología. Es decir, que dadas ciertas variables deberemos movernos en uno u otro sentido por la recta de la Figura 009.

Una variable muy importante a tener en cuenta es la cantidad de personas involucradas. Es algo natural pensar que a medida que aumenta la cantidad de personas asignadas a un proyecto deberemos contar con metodologías más pesadas tendientes a suplir la falta de canales anchos de comunicación entre los integrantes del equipo de desarrollo. Es por esto que la mayor parte de las metodologías ágiles dejan claro un tamaño máximo del equipo de desarrollo para utilizarlas exitosamente (en caso de tener equipos de desarrollo más grandes se recomienda particionarlos en equipos más pequeños). Alistair Cockburn [Cockburn, 2000] muestra este aspecto esencial a la hora de elegir una metodología mediante la Figura 010.

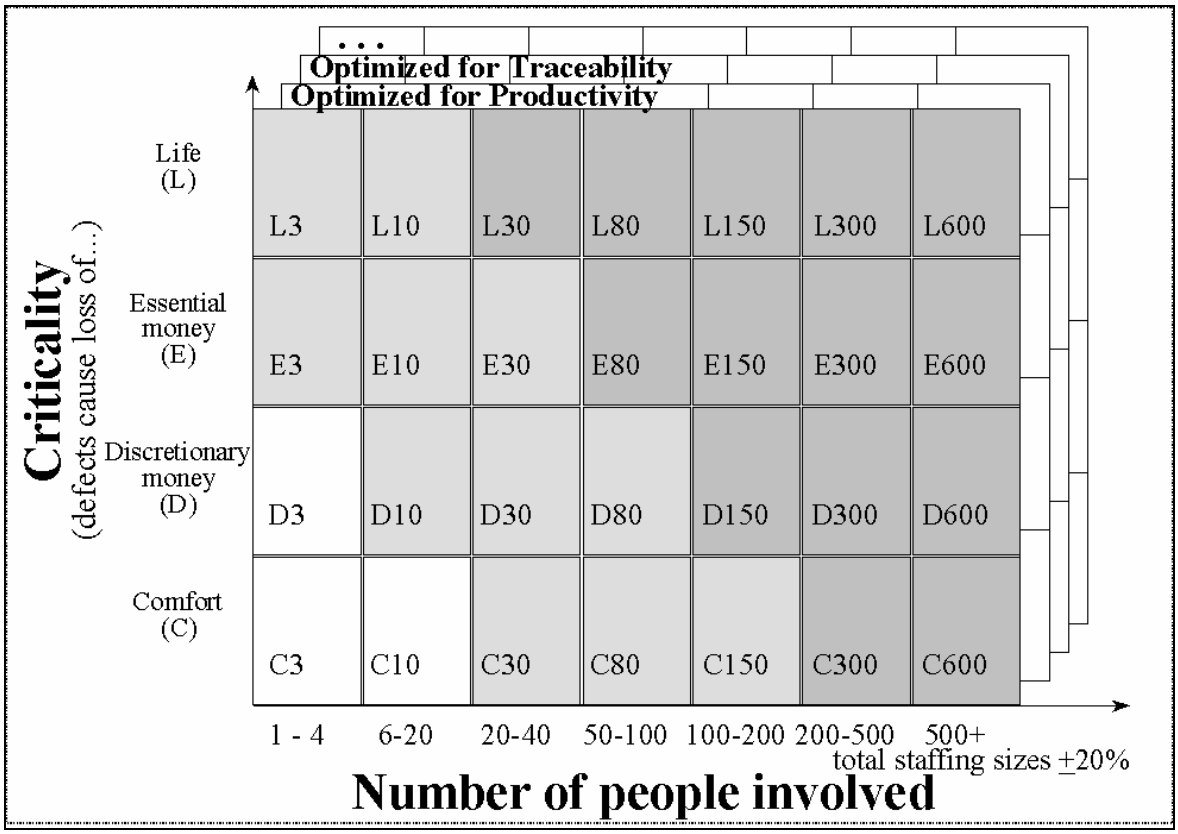

Figura 010. Relación entre cantidad de personas, criticidad, y tamaño de la metodología. Tomada de [Cockburn, 2001a].

AgEnD en particular está enfocada a pequeños grupos de desarrollo, de no más de 15 personas involucradas en el proyecto, las cuales deben compartir un ambiente de trabajo común, con mínima separación física. Cabe mencionar que en caso de contar con una mayor cantidad de recursos se deberán particionar en subgrupos del tamaño antes indicado, trabajando en forma paralela.

Otra variable a tener en cuenta, es el grado de criticidad de la aplicación. A medida que aumenta el perjuicio para el negocio ante una posible falla en el sistema, se deberá utilizar una metodología más pesada, con procesos de revisión bien pautados, de forma de asegurarse la calidad del producto. No es lo mismo construir una aplicación empaquetada para bajar archivos de internet automáticamente, que el software utilizado en un equipo de resonancia magnética. En el primer caso, una pérdida solo causa la posible pérdida de información para un usuario ocasional. En el segundo caso, una falla podría resultar en anomalías en los resultados obtenidos con la posibilidad de realizar diagnósticos erróneos. ¡Existe un riesgo de vida en relación a esta aplicación!

Aquí vemos la otra magnitud provista en la Figura 010 en la cual observamos que a medida que aumenta la criticidad, desde la pérdida de *Comfort* hasta la pérdida de

Vida, debemos agregarle más rigor a la metodología para estar seguros de entregar un software lo más confiable posible.

Es así que se postula AgEnD como una metodología para ser utilizada en proyectos de características C3, C10, D3, que tienen como máximo 20 personas en el equipo de desarrollo y que presentan un grado de criticidad bajo, de pérdida de *Comfort* o pérdida *Económica Discrecional*. Esta es la realidad de la mayoría de las pequeñas consultoras de desarrollo localizadas en Buenos Aires, que trabajan para los sectores de IS/IT.

Orientado al proceso vs. Orientado a las personas

Un par de décadas atrás, en el campo de la Ingeniería de Software solo existían los denominados procesos tradicionales. Comenzando por el modelo en cascada con su división en etapas y roles bien definidos, continuando por el modelo en espiral, donde se mitigaban los riesgos en forma temprana, todos estos procesos proponían un enfoque secundario en relación a las personas. Consideraban que teniendo un proceso predecible, con etapas y tareas bien definidas, el éxito estaba garantizado independientemente de las personas que cubrieran los roles. Estos modelos de procesos están orientados al proceso en si, y suponen que al seguir los principios propuestos por los mismos en cualquier organización y con cualquier equipo de desarrollo, se obtendrán los resultados predichos en la teoría. El resultado de aplicar un proceso de calidad, será la calidad en el producto.

Actualmente nos encontramos con esta visión orientada al proceso en muchas organizaciones con altos niveles de burocracia y en organizaciones basadas en el modelo Codificar y Probar. Es decir, en ningún momento existen consideraciones respecto a como la productividad se ve afectada por la gente que trabaja y por las condiciones laborales existentes. Existen referencias en múltiples libros respecto a como el factor humano es fundamental al éxito del proyecto [DeMarco, 1987][Weinberg, 1998][McConnell, 1996][McConnell, 2003][Glass, 2002]. Para evaluar la importancia de este elemento podemos tomar uno de los frameworks de estimación más importantes en la actualidad COCOMO II desarrollado por Barry Boehm entre otros [Boehm, 2000].

El método en cuestión propone siete factores de personal que ajustan la estimación de un proyecto. Los mismos están relacionados con:

- Experiencia en aplicaciones
- Factores comunicacionales
- Experiencia en el lenguaje y las herramientas
- Continuidad del personal
- Experiencia en la plataforma
- Capacidad de los programadores
- Capacidad de los analistas

Es decir que dependiendo de factores exclusivamente humanos nuestro proyecto puede variar en una relación de hasta un 24,6 – es decir un proyecto con todos los factores humanos negativos puede llegar a tardar hasta 24,6 veces más que un proyecto con un equipo de desarrollo adecuado y sin deficiencias. De ahí el énfasis de las metodologías ágiles en este tipo de cuestiones.

Contando con la experiencia de Alistair Cockburn en estos temas recolectada a lo largo de varios proyectos relevados en su mayoría en IBM, el paper denominado *Characterizing People as Non-Linear, First-Order Components in Software Development* [Cockburn, 1999] demuestra que las personas son componentes de primera magnitud en cualquier desarrollo. De esto surge uno de los principios fundamentales de todas las metodologías ágiles que se ve plasmado en uno de los primeros valores del [Manifesto, 2001]:

Individuos e Interacciones por sobre procesos y herramientas

El énfasis de todas las metodologías ágiles esta centrado en los individuos y sus interacciones, a diferencia de las metodologías tradicionales en que impera el proceso y las herramientas como propulsores de la calidad. AgEnD no es ninguna excepción a estos principios y propone destacar en cada una de sus prácticas que las personas son las

que indudablemente decidirán sobre la mejor forma de trabajo para ellas. También de las personas dependerá la adopción de AgEnD o cualquier proceso de desarrollo ágil. Es por esto, que más adelante se define un rol en particular, el Coordinador, que se encarga de mantener las prácticas de AgEnD en armonía, durante las primeras iteraciones en que se utiliza el proceso.

Consideraciones Humanas

El hecho que AgEnD base sus principios en las personas no significa que se han resuelto todos los problemas del desarrollo y que teniendo un buen equipo el éxito está garantizado. Todo lo contrario, el énfasis propuesto requiere tomar ciertas consideraciones en relación a la naturaleza de las personas que no son necesarias en las metodologías tradicionales. Partimos de la base formulada por [Cockburn, 1999] que las personas son organismos vivientes impredecibles, inconsistentes pero con buena voluntad y un sentido de hacer las cosas bien. De esto concluimos que la Figura 010 extraída del [RUP, 2002][2] establece ciertas suposiciones que han sido esenciales en las prácticas tradicionales de management basadas en la ideología de *Command and Control*.

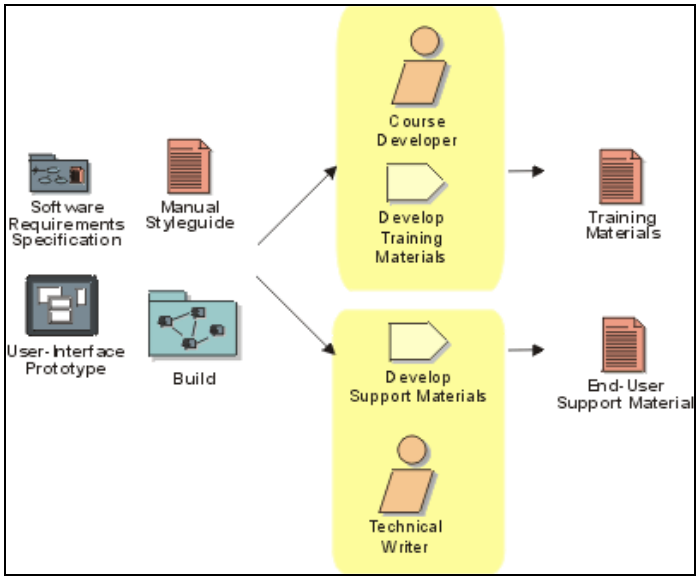

Figura 011. Descripción de un workflow dentro del RUP. Tomada de [RUP, 2002].

[2] La selección de la figura del RUP y el comentario posterior no intenta realizar ninguna referencia a la naturaleza de dicho framework.

Como se observa el *Desarrollador del Curso,* toma como inputs la *Especificación de Requerimientos Suplementarios*, la *Guía de Estilo del Manual, el Prototipo de la Interfase de Usuario* y el *Build* correspondiente. Con esto realiza la tarea *Desarrollo de Materiales de Entrenamiento*, y genera el output *Materiales de Entrenamiento*. En el segundo caso, es el *Escritor Técnico* el que toma los mismos inputs, pero esta vez realiza la tares *Desarrollo de Materiales de Soporte,* generando el output *Material de Soporte para el Usuario Final*.

Es en esta figura en que se puede analizar la ideología de las metodologías tradicionales. Uno parte de ciertos artefactos como entrada, define los roles involucrados, define la tarea a ser realizada, y finalmente, se obtienen los artefactos deseados. La definición de las tareas es realizada en bastante detalle, y se suele utilizar la técnica del WBS para particionar un proyecto en las tareas que lo componen. Si bien las metodologías ágiles parten de figuras similares, tienen en cuenta la naturaleza de las personas y consecuentemente analizan formas en que las personas obtienen el mejor output a partir de dichos inputs.

Uno de los elementos fundamentales asociados a las personas y las interacciones es el *Equipo de Desarrollo*. Dado el incremento continuo en la complejidad del software, no alcanza con esfuerzos individuales para tener éxito. Se deben contar con equipos productivos que pueden construir software en forma iterativa, en los que exista un alto grado de comunicación y donde cada integrante se sienta parte de una comunidad de personas con un interés en común: construir software de calidad. Estos equipos de desarrollo deberán cubrir dos aspectos que según [Royce, 1998] definen la excelencia de un equipo: *balance* y *cobertura*.

Orientado a los productos vs. Orientado a las tareas

Otro punto de inflexión que se generó con el movimiento de las metodologías ágiles parte de la importancia de los denominados timeboxes o iteraciones. Los mismos indican una clara tendencia a focalizarse en los productos a ser entregados en cada iteración, restando importancia a las tareas necesarias para obtener los correspondientes entregables.

Realizando una analogía con el testing, se podría considerar que AgEnD aplica lo que denominaremos desarrollo de *caja negra* mientras que las metodologías tradicionales utilizan el desarrollo de *caja blanca*. El primero refiere a la noción de tomar ciertas entradas y verificar las correspondientes salidas. El segundo refiere a tomar ciertas entradas, analizar el proceso interno y verificar las correspondientes salidas. Las metodologías tradicionales al considerar a las personas como partes reemplazables de un desarrollo, definen en detalle entradas, salidas y tareas necesarias para llevar a cabo la correspondiente transformación. Consecuentemente, estas metodologías proponen que basta con tener una clara definición de las tareas en un proyecto para garantizar el éxito del mismo. Un WBS nos provee la seguridad necesaria para avanzar en las sucesivas etapas, en el cual describiremos que tareas son necesarias, que necesita cada una para comenzar, y con que artefactos finalizan. Considerando a las personas como componentes predecibles y reemplazables, engranajes de una maquinaria, el desarrollo de *caja blanca* se convierte en la forma más adecuada de llevar un proyecto al éxito en organizaciones orientadas al proceso. Ejemplos de esto se observan en los proyectos de software encarados por el departamento de defensa y aquellas grandes organizaciones en las que se tienen niveles de madurez certificados por el CMM.

Una de las razones principales por la que las metodologías ágiles son Orientadas a los Productos es el principio de *Máxima Comunicación* – descrito en más detalle en las Características de AgEnD. Dada la necesidad de tener un feedback continuo por parte del cliente, el énfasis de cualquier iteración esta puesto en los artefactos que esta genera. Las actividades necesarias para la transformación de entradas en salidas queda a discreción del equipo de desarrollo pudiendo utilizar las técnicas provistas por AgEnD que proporcionan agilidad al proceso. Para lograr que el desarrollo de *caja negra* tenga éxito deberemos tener un equipo integrado, balanceado y altamente motivado. De estos tres factores, es la motivación la que nos permitirá lograr mejores resultados en la adopción de un proceso ágil. Es la motivación la que logra sacar lo mejor de un equipo de desarrollo; la que nos permite saber que las personas involucradas utilizarán las técnicas que les sean más efectivas para llegar al final de la iteración con los artefactos más críticos construidos de acuerdo a las estimaciones. La motivación es por lejos la mayor contribuyente a la productividad [Boehm, 1981].

Desarrollo Iterativo

Dado el avance de la tecnología, los tiempos en que se manejan los negocios han ido reduciéndose abismalmente. Lo que una década atrás demoraba una semana de procesamiento computacional, hoy se resuelve en cuestión de minutos. Los tiempos de procesamiento han ido disminuyendo, mientras que los sistemas han ido creciendo en complejidad teniendo cada vez necesidades mayores de distributibilidad, seguridad, mantenibilidad, flexibilidad, etc.

En el panorama actual en que se encuentra la industria de IS/IT los sistemas deben contar con atributos jamás pensados en sistemas tiempo atrás. Palabras como: multiprocesamiento, sistemas distribuidos, middleware, web services, modelo en tres capas, tolerancia 24x7, multiplataforma, Virtual Machine, son requerimientos no funcionales típicos de sistemas de software/hardware en la actualidad. Los “nuevos” clientes de los actuales sistemas esperan tener su sistema en el menor tiempo posible, con la máxima calidad y conforme a los requerimientos no funcionales necesarios para la operatoria diaria del negocio.

Una de las tareas más complejas en el desarrollo de sistemas representa la determinación del alcance de lo que se va a construir. Es decir, qué es lo que recibirá el cliente en cada release del producto. ¿Que funcionalidad estará incluida? ¿Cómo asegurarnos que es esa la funcionalidad que resuelve las necesidades del cliente? Existen innumerable cantidad de ejemplos en la industria de sistemas de proyectos con extensos períodos de relevamiento, que fueron seguidos por también extensos períodos de “oscurantismo” durante los que el cliente no tenía feedback alguno del desarrollo. En general, el resultado de estos proyectos era el fracaso ya que la aplicación no cumplía con las necesidades del usuario. El síndrome del IKIWISI (I’ll Know It When I See It) entraba en juego, y al no tener feedback alguno del lado del cliente la construcción se llevaba a cabo sobre las especificaciones definidas en forma temprana.

Si tomamos la teoría existente detrás del Modelo en Cascada, la misma no hacía más que reflejar las teorías encontradas en las otras ingenierías. En estas últimas, los proyectos comenzaban con fases de ingeniería en que se constataba la factibilidad de los proyectos, se sentaban las bases para la producción y se mitigaban los posibles riesgos que podían ocurrir. Una vez que se tenía una cierta masa crítica de conocimiento, se

realizaba una transición a la fase de producción. En la misma, se ejecutaban todas las tareas analizadas y diseñadas en las fases anteriores, las cuales eran de carácter predecibles y contaban con mayor cantidad de RRHH de menor nivel de capacitación, y menores ingresos.

Dadas las características de dichas ingenierías (Civil, Industrial, Hidráulica, Química, etc.) las mismas se prestaban a procesos de desarrollo de características predecibles. Tenían un conjunto de requerimientos, los cuales se mantenían estáticos durante la ejecución y no se modificaban en forma considerable hasta el último día de producción. En estos principios se basa la industria manufacturera, en la cual una fase de desarrollo inicial prescribe todas las actividades a realizarse en la ejecución. La predictiblidad es alta y la variabilidad es mínima.

Sin embargo esto mismo no aplica al dominio del software el cual tiene correlatos más fuertes con el desarrollo de nuevos productos. En la década del ’70 ya se proponían enfoques que mitigaban los problemas de intentar encajar el desarrollo de software bajo un esquema de línea de producción. Los problemas con la integración realizada al final del proyecto fueron solventados mediante el desarrollo iterativo.

Mientras que la curva de costos del software demostraba que a medida que se avanzaba en el desarrollo iban aumentando los costos de cambiar e incluir nuevas funcionalidades en forma exponencial, fue solventado mediante el énfasis de las metodologías del momento en la ingeniería de requerimientos, el análisis funcional y el diseño de componentes.

El énfasis de los mismos plantea la necesidad de una continua priorización de funcionalidad a ser desarrollada en cada iteración. El mantener fija una de las variables del proyecto permite jugar con las demás variables en forma dinámica.

Capítulo III - Solución Propuesta

Se ha descrito en capítulos anteriores en qué consiste un modelo de proceso de desarrollo de software, cuál es su propósito y las ventajas que trae aparejada la definición y utilización de uno dentro de una organización. Más aún, se ha realizado una extensa investigación en relación a las metodologías ágiles más destacables en la actualidad presentando un resumen de cada una y analizando el contexto de aplicación en que son utilizadas.

En este capítulo se verá en detalle cómo está estructurada AgEnD, la metodología ágil que da una solución al problema del desarrollo de software de alta calidad. En particular la misma fue concebida con la idea de ser aplicada en pequeños grupos de desarrollo, sin necesidad de invertir en costosas herramientas o de capacitar en forma extensiva a las personas involucradas.

Por tratarse de una metodología ágil se intenta minimizar la burocracia que subyace en la utilización de complejos procesos determinísticos que son utilizados en proyectos con gran cantidad de recursos. Es por esto que se han anexado en este capítulo aquellos templates que pueden ser utilizados para que el equipo de desarrollo estandarice los entregables que generará. Esto permitirá que la transición hacia la metodología sea lo menos dolorosa posible, logrando disminuir la curva de aprendizaje inherente a un cambio de proceso.

Características de la Metodología

Para dar una primera aproximación a AgEnD nos basaremos en la propuesta de Alistair Cockburn [Cockburn, 1998] que desglosa una metodología en 10 elementos, como mínimo, enumerados a continuación:

1. **Roles** – las descripciones a los trabajos que se solicitan en las búsquedas laborales cuando se necesitan tomar gente; por ejemplo: moderador JAD, project manager, arquitecto, escritor, tester.

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

En particular en este capítulo nos referiremos a los primeros siete puntos de la lista mencionada por tratarse de cuestiones estructurales que permiten describir el conocimiento asociado con AgEnD. Respecto a los demás puntos podemos decir que:

- AgEnD no prescribe técnicas salvo en algunos casos en particular en que son mencionadas en conjunto con las actividades correspondientes. Esto está dado por la *Orientación a las Personas* y *a los Productos* ya descrita.
- AgEnD no prescribe ningún tipo de herramienta para llevar a cabo las actividades. Sin embargo, en algunos casos se mencionarán ciertas herramientas de carácter Open Source pero meramente como ejemplificación, sin ninguna connotación comercial asociada.

- AgEnD no prescribe estándares de ningún tipo para los entregables. La única referencia a esto es el Anexo de Templates al final de la tesis, que solo intenta ser una guía respecto a cómo se pueden construir ciertos artefactos.

Roles definidos

Una de las razones principales de la adopción de una metodología para desarrollo consiste en la definición de las tareas que serán llevadas a cabo por los individuos que participan del proyecto. Los roles definen ese conjunto cohesivo de actividades realizadas y artefactos mantenidos que son llevados a cabo por personas o por grupos de personas. No sólo se refieren a personas internas al desarrollo del software, sino que también involucran a los usuarios u otras personas que se vean afectadas por el proyecto, cualquiera de los denominados *stakeholders*.

Los roles recién mencionados no son roles exclusivos asociados a una única persona, aunque en algunos casos pueden serlo (Patrocinante). Algunos roles podrán ser abarcados por más de una persona (Desarrollador). Asimismo, una persona podrá cubrir más de un rol (Arquitecto/Escritor Técnico).

Las asignaciones serán llevadas a cabo por el Líder, el cual en base a las aptitudes de los recursos que dispone repartirá las tareas a ser realizadas en cada iteración.

Como punto de partida en la definición de roles dentro de AgEnD, enumeraremos brevemente cada uno, con una descripción concisa de las actividades que abarcan:

- Patrocinante (Executive Sponsor)

  Actividades: tiene a su cargo el soporte gerencial del proyecto; es el encargado de proveer, comunicar y mantener actualizada la Visión del proyecto; provee el presupuesto para la viabilidad económica del desarrollo; es responsable por la consecución del proyecto del lado del cliente.

Importancia del rol: es esencial para el éxito del mismo, ya que un software que no tiene aceptación dentro de la organización que lo financia jamás llegará a ser construido en tiempo, forma y con consentimiento de los usuarios, no siendo utilizado eventualmente si se concreta el proyecto.

- Líder de Proyecto (Project Manager)

  Actividades: tiene a su cargo la planificación del proyecto, a lo largo de todo el ciclo de vida, incluida la planificación en detalle de cada iteración; asigna recursos y delega responsabilidades en los mismos; fomenta la cohesión del grupo y lleva a cabo actividades destinadas a eliminar fricciones; organiza las reuniones a ser realizadas; monitorea el progreso del proyecto y establece estrategias para mitigar los riesgos que se puedan presentar.

  Importancia del rol: el Líder de Proyecto representa la cara visible del equipo de desarrollo, es el nexo existente entre la gerencia y el equipo de desarrollo como se observa en la Figura 012.

- Experto en el Dominio (Domain Expert)

  Actividades: tiene a su cargo brindar su conocimiento del negocio contribuyendo al modelado del sistema que llevan a cabo los Analistas durante la disciplina de Requerimientos-Análisis; participará junto con los Testers en la definición del contenido de las pruebas funcionales a ser realizadas; será el responsable de la aprobación de las pruebas de aceptación por cada release entregado.

  Importancia del rol: el Experto en el Dominio permite al Equipo de Desarrollo aprender sobre el negocio para el cual está siendo construida la aplicación; son encargados de resolver cualquier cuestión relacionada con la funcionalidad de la aplicación junto con los Analistas.

Destrezas: el Experto en el Dominio deberá conocer en detalle el negocio para prestar respuesta a cualquier duda que pueda surgir del mismo. En general será un miembro de la empresa Cliente.

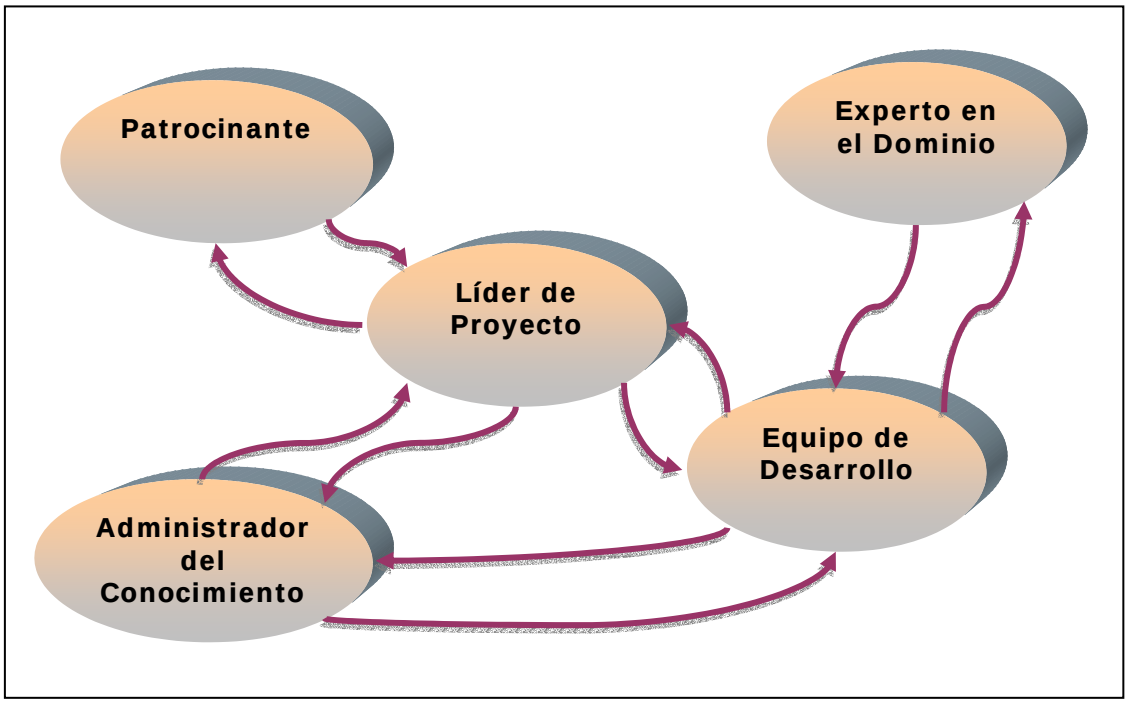

Figura 012. Flujos de comunicación durante el proyecto entre los roles propuestos.

- Coordinador (Mentor)

  Actividades: tiene a su cargo la supervisión del proceso, y cualquier actividad orientada al mejoramiento del mismo. Durante las primeras etapas de utilización de AgEnD supervisará la implementación del proceso.

  Importancia del rol: en las metodologías ágiles este rol permite reforzar la adherencia al proceso en aquellos momentos en que se el tiempo apremia y se suele caer en el modelo *Codificar y Probar*.

  Destrezas: el Experto en el Dominio deberá conocer en detalle el negocio para prestar respuesta a cualquier duda que pueda surgir del mismo.

- Analista (Functional Analyst):

Actividades: tiene a su cargo el relevamiento, mediante el cual se obtienen los requerimientos de la aplicación a ser construidos en cada iteración; realiza la especificación de los requerimientos; prepara el documento de Visión.

Importancia del rol: el aprendizaje del dominio de la aplicación y de los requerimientos que deberá tener la misma son claves para el éxito del proyecto y la aceptación del mismo por parte del usuario.

Destrezas: el Analista deberá tener amplio conocimiento de técnicas de relevamiento, así como aptitudes sociales que le permitan vencer el "Síndrome del Usuario y el Desarrollador" [Leffingwell, 2001].

- Arquitecto (Architect):

Actividades: tiene a su cargo la definición de la arquitectura que guiará el desarrollo, y de la continua refinación de la misma en cada iteración; deberá construir cualquier prototipo necesario para probar aspectos riesgosos desde el punto de vista técnico en el proyecto; definirá los lineamientos generales del diseño y la implementación.

Importancia del rol: la arquitectura es imprescindible en los proyectos de software actuales en donde cada vez existe mayor complejidad; el arquitecto puede ser considerado como el Experto en la parte técnica del desarrollo y debe mantener a todo el equipo en conocimiento de los lineamientos fundamentales de la construcción.

Destrezas: el Arquitecto deberá tener una buena formación técnica, contar con experiencia en las herramientas y técnicas utilizadas; aptitudes comunicacionales son deseadas para que la arquitectura sea comunicada a todos los miembros del equipo; también deberá ser perseverante en conseguir los hitos técnicos planteados mediante entregables para asegurar el progreso de la construcción.

- Programador o Desarrollador (Designer-Programmer):

Actividades: tiene a su cargo la codificación de los componentes a desarrollar en la iteración; debe crear y ejecutar los tests unitarios realizados sobre el código desarrollado; es responsable de las clases que ha desarrollado debiendo documentarlas, actualizarlas ante cambios y mantenerlas bajo el control de configuración de las mismas mediante la herramienta de SCM utilizada.

Importancia del rol: el Programador es la persona que tiene los materiales y lleva a cabo la implementación de los casos de uso en el lenguaje de programación elegido; como en general, es el paradigma de objetos[3] el que está imponiéndose en la industria de IS/IT, el Programador definirá las clases y métodos que realicen los correspondientes casos de uso.

Destrezas: el Analista deberá tener amplio conocimiento de las herramientas de desarrollo, del lenguaje de programación, de los aspectos técnicos involucrados.

- Tester:

  Actividades: tiene a su cargo la generación de pruebas funcionales a partir de los requerimientos extraídos por los Analistas.

  Importancia del rol: la importancia del Tester radica en la necesidad de construir un software de calidad que cumpla con los requerimientos del usuario; mediante la utilización de un proceso y el armado de un grupo cohesivo de desarrollo, se tienen prácticas para garantizar la calidad en el producto desde el punto de vista técnico. Sin embargo, para asegurarnos de que la aplicación satisface las necesidades del usuario deberemos realizar todo tipo de pruebas de carácter funcional. Es ahí en que entra en juego el Tester, quien crea, ejecuta, analiza y mantiene el conjunto de pruebas automatizadas y manuales que son utilizados.

  Destrezas: el Tester deberá tener amplio conocimiento de técnicas de testing, deberá conocer a fondo la aplicación que testeará. Asimismo,

deberá tener conocimientos de programación para trabajar con las pruebas automatizadas.

- Administrador del Conocimiento (Knowledge Manager):

  Actividades: tiene a su cargo la captura, refinamiento, empaquetamiento, y transferencia del conocimiento, ya sea tácito o explícito, en la organización. En particular, AgEnD recomienda que este rol sea llevado a cabo por una persona del equipo de desarrollo con dedicación part-time – esto puede depender del tamaño de la organización y de otros factores.

  Importancia del rol: el Administrador del Conocimiento es uno de los pilares sobre los que se establece AgEnD. Su importancia consiste en la capacidad del equipo de desarrollo de aprender de la experiencia que éste y que otros equipos dentro de la organización generan a diario durante el transcurso de los proyectos. Mediante esta disciplina se logra el reuso de dicho conocimiento. Este tema se trata con más detalle en la sección "Administración del Conocimiento".

  Destrezas: el Administrador del Conocimiento debe poseer aptitudes en comunicación para poder capturar el conocimiento de aquellas personas que lo generan.

Aptitudes Requeridas

Uno de los principios esenciales que mencionamos al principio de esta tesis, era el de *Máxima Comunicación*. Este principio establecía la necesidad de un continuo aprendizaje del equipo de trabajo, que debía ser transmitido en forma rápida a todos los involucrados.

Ahora bien, algo que estaba implícito en la enunciación del principio era el hecho de disponer de recursos *dispuestos y capacitados para aprender*. No alcanza con sentar a todas las personas en una misma habitación para lograr la fluida comunicación

[3] Como se observa por estudios diversos de Gartner, Forrester, entre otros. Las dos plataformas que eventualmente comprenderán el mayor porcentaje de los futuros desarrollos, Java y .NET, están construidos sobre lenguajes orientados a objetos los cuales basan su

típica de los procesos ágiles. También es necesario que esas personas estén dispuestas a sacrificar su individualidad en pos de un grupo cohesivo de desarrollo. De esta forma, se logra un entendimiento que abarca extensamente el alto grado de complejidad inherente al software.

Suponiendo una situación hipotética en la que una empresa ha conseguido un proyecto a ser implementado utilizando AgEnD, y debe contratar a todos los recursos para el mismo, mencionaremos qué aptitudes serían necesarias según los roles propuestos por la metodología.

Patrocinante

En relación al Patrocinante, deberemos requerir simplemente un total e incondicional apoyo al proyecto, debiendo negociar siempre a favor de la construcción del software y su implantación en la empresa. En este sentido, no se agrega nada en particular al conocimiento planteado por la bibliografía existente sobre Ingeniería de Software.

Experto en el Negocio

El Experto en el Negocio debe ser una persona con amplio conocimiento del dominio de la aplicación que pueda concebir una visión lo suficientemente amplia del producto a ser entregado para poder guiar el proceso de desarrollo. No será necesario que el experto en el dominio sepa en detalle toda posible operatoria del sistema; sin embargo, deberá contar con el suficiente conocimiento para poder resolver cualquier duda o dificultad que tengan los miembros del equipo de desarrollo respecto a los requerimientos. También deberá contar con la autoridad requerida para definir prioridades respecto a los casos de uso a ser implementados en cada iteración, pudiendo agregar funcionalidad que no estaba contemplada o descartando funcionalidad que perdió valor para el negocio.

Existirá una estrecha relación entre el Experto en el Negocio y los Analistas, encargados de realizar los casos de uso. Durante las iteraciones del desarrollo, se necesitará una total disponibilidad del Experto para la redacción de los casos de uso así como una frecuente predisposición a contribuir en el desarrollo subsecuente.

comportamiento en clases, atributos y métodos.

Será necesario capacitar al Experto en la metodología para que se pueda ajustar mejor al ritmo del desarrollo y contribuya reforzar la cohesión del grupo de trabajo. Para esto, será necesario que no tenga problemas en relacionarse con las demás personas involucradas.

Líder

En el caso del Líder, deberíamos contratar a una persona que:

- explote las aptitudes individuales de los integrantes del equipo, de forma de sacar lo mejor del grupo
- tenga muy en claro las características de los procesos iterativos
- confíe en el grupo de trabajo y delegue sin ningún tipo de temor
- comprenda que se encuentra en un proceso de aprendizaje continuo, y que los errores son parte del trabajo
- trate de minimizar las fricciones existentes en el grupo
- confíe plenamente en la metodología y no la deseche a la primer señal de desviación
- no sufra de omnipotencia, tratando de regir estrictamente el trabajo de las personas bajo su cargo

Teniendo en cuenta las estructuras organizacionales de las grandes empresas, observaremos que el Líder requerido para los proyectos realizados utilizando AgEnD corta con el estereotipo del típico Líder de Proyecto. No sólo estamos hablando de cambios de actitud y comportamiento, se plantea un cambio de mentalidad radical que consiste en dejar de lado el aspecto determinístico del software y convencerse de la complejidad esencial que posee. Se debe pensar en el cambio como una fase más del software, la cual sobreviene en forma dispersa durante las iteraciones, y que nos sirve para aprender más sobre lo que estamos construyendo y para poder cubrir mejor las necesidades de los usuarios.

Si bien estas características deben ser comprendidas por toda persona que participe en el proceso, es esencial que sean asimiladas por el Líder ya que éste deberá encauzar nuevamente al equipo para que no ceda al caos total en el desarrollo.

Coordinador

El Coordinador es aquella persona que esta íntimamente ligada con el proceso, que conoce todas las prácticas involucradas y entiende el porqué de los principios de la misma. AgEnD propone que el énfasis debe centrarse en las personas en vez del proceso, lo cual parecería contradecir la existencia del Coordinador. Pero esto no es así ya que en caso que no exista un responsable del proceso, dada la naturaleza ágil del mismo, se puede terminar fácilmente en el modelo de codificar y probar.

Su tarea consiste en reforzar la convicción de la necesidad de continuar con las prácticas y principios de AgEnD aun en los momentos más críticos del proyecto. El equipo de desarrollo debe estar convencido plenamente de que las prácticas optimizan el desarrollo, y no que resultan un obstáculo del que deben deshacerse rápido. Es en estos momentos en que se ve la importancia del Coordinador. Otra circunstancia en la que resulta útil su presencia, es ante la incorporación de nuevos recursos, a los que el Coordinador capacitará para que puedan ajustarse rápidamente sin que esto perjudique el normal funcionamiento del grupo. Una vez que el grupo de trabajo se siente ya cómodo con la metodología, el Coordinador puede desaparecer como tal y ser reemplazado por algún miembro del equipo.

Analista

El Analista cumple un rol similar a la figura de Analista Funcional con la que se define este empleo en la industria del software. Es el encargado de realizar el relevamiento, identificando previamente a los actores stakeholders que deberá relevar para entender el dominio del problema. En dicho relevamiento se incluirá al Experto en el Negocio, los Usuarios y demás stakeholders del proyecto, y posteriormente se plasmarán los requerimientos funcionales y no funcionales en especificaciones de casos de uso.

Una de las aptitudes necesarias del Analista es la de entender a los Usuarios, la de ser una persona lo suficientemente humilde como para reconocer que no tiene experiencia en el dominio y que debe pedir que se le explique cómo es la normal operatoria del negocio para poder capturar su esencia. El Analista es una persona versada en tópicos de Ingeniería de Software, con amplios conocimientos de técnicas de relevamiento, especificación de requerimientos, y manejo de cambios, que posee bastante conocimiento sobre el negocio en el que opera el Cliente, quien financia el

proyecto[4]. En respuesta a esto AgEnD mantiene con firmeza un principio que consiste en uno de los pilares de las metodologías ágiles, el de *Máxima Comunicación*.

Los requerimientos cambiarán antes, durante e inclusive una vez finalizado el proyecto.

Esta es una dimensión característica al software: a medida que se avanza en el desarrollo, el Cliente va teniendo una noción más definida de qué es lo que quiere, lo que produce un gradual cambio de visión que se traduce en una continua mutación de requerimientos. El Autor propone identificar esta situación como el Síndrome de la Meta Difusa (Fuzzy Goal Syndrome). Dado que la meta no es conocida de antemano, no tiene sentido empeñar gran cantidad de recursos al principio del desarrollo en metas difusas.

Esta aptitud se deberá ver reflejada en el Analista, y en el mejoramiento continuo de su entendimiento. El análisis iterativo contribuirá a que el producto entregado satisfaga los requerimientos que tenía el usuario hasta ese momento, pudiendo incluirse cambios en sucesivos releases.

Arquitecto

El arquitecto es aquella persona encargada de dirigir y coordinar los aspectos técnicos del desarrollo, proveyendo de coherencia a los diversos artefactos generados en relación al análisis, diseño, implementación, testing y despliegue. Deberá estar comunicado con el resto del equipo para transferir los conceptos arquitectónicos más relevantes a aquellos que los necesiten, y al mismo tiempo para absorber los detalles técnicos que deberán ser incluidos en la arquitectura. No será necesario que conozca en detalle cada componente construido, o documento generado, ya que para ello están las personas responsables de los mismos. Si deberá mantener una visión general de las partes técnicas del desarrollo, permitiendo de esta forma ir ajustando la arquitectura candidata de acuerdo a lo que la implementación de los casos de uso revele.

Algunas aptitudes básicas que se pedirían son:

- pleno conocimiento de las tecnologías disponibles
- manejo de notaciones estándar (UML) para comunicación entre las partes

[4] Cabe mencionar que si esta condición no se da, implicará un alto riesgo para el proyecto ya que la persona encargada de especificar lo que el sistema debe realizar no conoce los procesos del negocio. Una forma de mitigar este riesgo es mediante la realización de una disciplina de Modelado del Negocio [RUP, 2002].

- buena capacidad de comunicación con las personas del equipo
- contar con experiencia en el dominio del problema
- liderazgo para conducir a el o los equipos involucrados transmitiendo las decisiones técnicas tomadas, trabajando conjuntamente con el Líder

Programador

El Programador es aquella persona que tiene a su cargo la codificación de los componentes en código fuente en algún lenguaje de alto nivel, y la correspondiente prueba unitaria de los servicios que los mismos proveen. Para esto deberá tener un profundo conocimiento del lenguaje, las herramientas utilizadas en la construcción, los estándares de codificación, el framework de testing utilizado, patrones de diseño a aplicar y nociones de refactoring.

El Programador es la persona encargada de la construcción del sistema propiamente dicha. Deberá ser capaz de estimar la duración de las tareas que le son asignadas, incluyendo el testing unitario de las clases que genera. Si bien al principio pueden resultar imprecisas las estimaciones dentro de AgEnD, con el tiempo los Programadores se acostumbrarán al ritmo ágil que imprime el proceso pudiendo lograr gran precisión en las mismas.

Tester

Dada la importancia del testing en el proceso como forma de asegurarse que la calidad forme parte del producto, deberemos contar con una persona dedicada a verificar los requerimientos del usuario mediante casos de prueba que deberán ser creados en forma paralela con el desarrollo.

Si bien una parte del testing de AgEnD será realizada por los Programadores en forma de pruebas unitarias sobre el código construido, la porción de testing más relevante para las partes involucradas se refiere a las pruebas funcionales. Y estas deberán ser construidas por el Tester que deberá tener una estrecha comunicación con el Cliente ya que será éste quien defina el contenido de las mismas.

Entre otras cosas el Tester deberá contar con el suficiente conocimiento técnico como para poder entender la arquitectura del sistema, de forma de generar el código fuente de las pruebas funcionales necesarias para el sistema. Tomando como base los casos de uso generados por los Analistas su función será la de transformarlos en casos

de prueba. Estos casos de prueba estarán compuestos por dos componentes principales: código fuente y documentación. Para el primer enfoque se llevarán a cabo programas que verificarán la correcta realización de los casos de uso vía la interacción de las clases que lo realizan. El segundo enfoque refiere a documentar procedimientos de prueba, que consisten en instrucciones técnicas necesarias para probar un componente.

Administrador del Conocimiento

El Administrador del Conocimiento deberá tener conocimiento sobre las distintas áreas de la ingeniería de software de manera de poder empaquetar el conocimiento en forma estructurada.

Este rol deberá contar aptitudes de síntesis y claridad de en la especificación de la información consolidada. También deberá ser capaz de capturar el conocimiento generado por el equipo durante el desarrollo sin influenciar negativamente en la productividad de la gente.

Técnicas

Las técnicas propuestas por AgEnD son técnicas que han sido adoptadas por la industria para atacar la complejidad de las distintas fases del desarrollo. Las mismas serán mencionadas brevemente; en caso que el lector desee profundizar en alguna podrá hacerlo a través de la bibliografía propuesta al final de la tesis.

Dadas las características de los proyectos ágiles, definiremos las técnicas que más se ajustan a esta filosofía y que colaboran al desarrollo de las tareas que son encaradas en cada iteración. Debido a esto, no se mencionarán en este resumen técnicas que si bien han sido probadas durante los años son de dudosa trazabilidad en relación a las demás fases, o bien generan un overhead demasiado costoso para proyectos de estas magnitudes.

Las técnicas propuestas serán mencionadas en relación a las fases en que se divide AgEnD.

Fases e Hitos

Como cualquier metodología AgEnD se compone de un conjunto de fases que son realizadas a lo largo del tiempo, las mismas conforman un período de tiempo encuadrado entre hitos significativos para el proyecto. Las mismas indican el énfasis ya propuesto por Barry Boehm en relación a la división en dos perspectivas de desarrollo. La primera perspectiva ocurre en las primeras iteraciones del proyecto, cuando se analiza la factibilidad de la ejecución del mismo y se termina con la obtención de la arquitectura con los riesgos más críticos mitigados. La segunda perspectiva ocurre en fases iterativas de construcción en donde el énfasis esta en el código fuente generado. Esto significa que el grado de creatividad requerido en el proceso disminuye con el tiempo, tornándose en un ciclo más predecible. En relación a la definición de fases e hitos de AgEnD se han tomado dos fuentes importantes [Boehm, 1995] y [RUP, 2002].

De acuerdo a la Figura 013, se observa la primera fase denominada *Concepción* que consiste en la definición de los features que tendrá la aplicación, el alcance de la misma, la identificación de los stakeholders, la customización del proceso a ser usado y llevar a cabo la planificación general del proyecto. Esta fase provee la fundación en que todo el posterior trabajo es basado. Es crítico en esta fase llevar a cabo los primeros relevamientos que se realizan con el Cliente de forma de adquirir conocimiento del dominio y analizar si los costos del proyecto estarán justificados o bien conviene comprar algún software empaquetado o cancelar la ejecución. La fase de *Concepción* finaliza con un hito mayor denominado **Objetivos del Ciclo de Vida** en el que se evalúa lo realizado contra las expectativas del Cliente y del Equipo de Desarrollo.

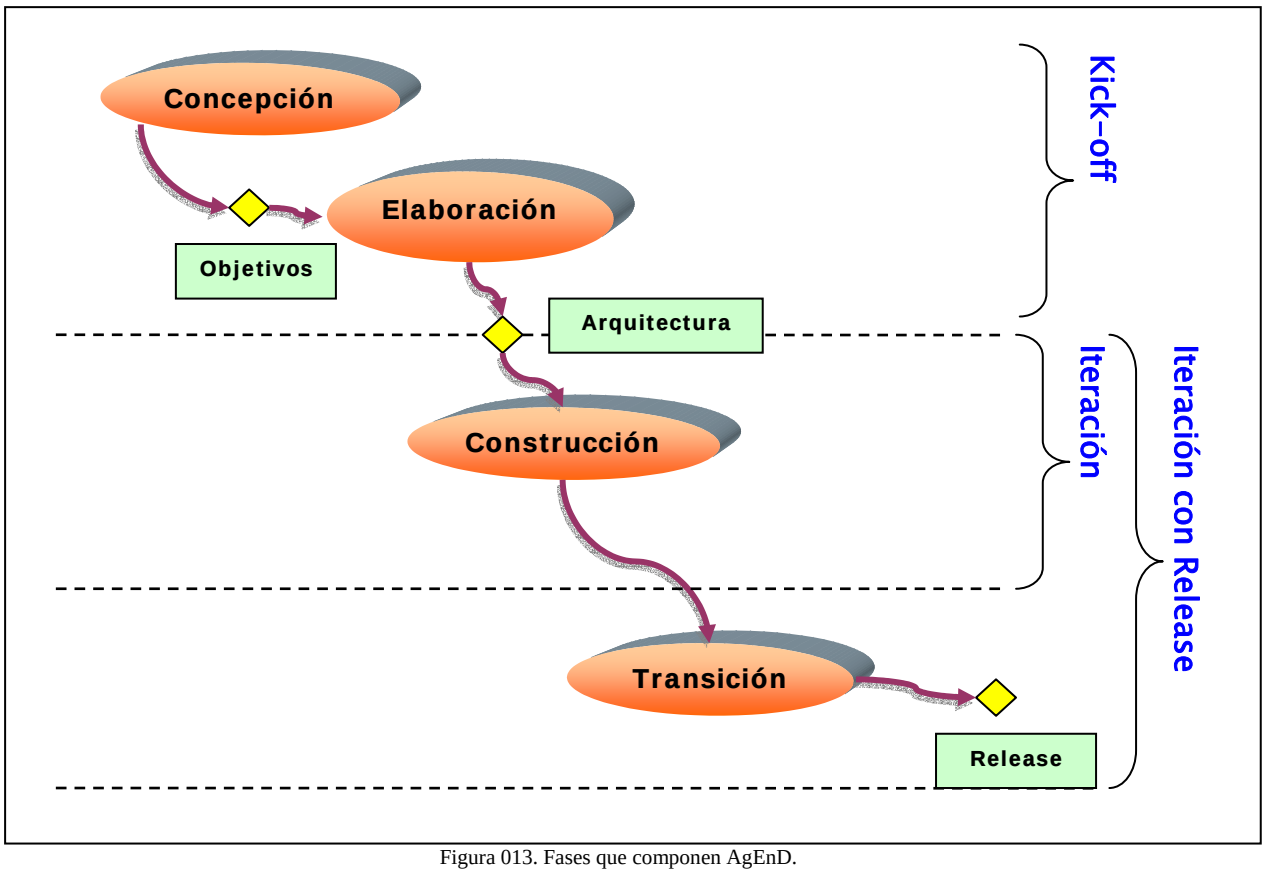

Figura 013. Fases que componen AgEnD.

La siguiente fase denominada *Elaboración*, se refiere a la exploración de los requerimientos más críticos (funcionales y no funcionales) que involucra el proyecto, así como las decisiones técnicas más importantes que quedarán plasmadas en el documento de Arquitectura. El objetivo principal consiste en asegurar la factibilidad técnica respecto a la realización del proyecto. Es a partir de la próxima fase cuando se comienza con la construcción a gran escala del software, en donde se comprometen la totalidad de los recursos necesarios para que el desarrollo se complete en las iteraciones planificadas. Asimismo, se podrán incorporar recursos en forma limitada tratando de no obstaculizar el normal transcurso del proyecto debido a la capacitación que estos últimos requerirán. La fase de *Elaboración* finaliza con un hito mayor denominado **Arquitectura del Ciclo de Vida** en el que se evalúa lo realizado contra las expectativas del Cliente y del Equipo de Desarrollo.

Una vez que pasamos a las etapas de producción tenemos dos fases concatenadas que se realizan en forma repetitiva por cada release de la aplicación. Estas fases son *Construcción* y *Transición*. Durante la *Construcción* se terminan de especificar los casos de uso correspondientes a la iteración, se diseñan los mismos bajo

la arquitectura candidata presentada, y se codifican todos los componentes definidos por los casos de uso, ejecutándose el testing correspondiente y la integración. Cuando se quiera pasar un cierto conjunto de componentes al entorno productivo se tendrá una fase de *Transición* en la cual se llevarán a cabo las actividades de despliegue necesarias (ver Figura 013 con Iteración con Release). La fase de *Transición* finaliza con un hito mayor denominado **Release del Producto** en el que se evalúa lo realizado contra las expectativas del Cliente y del Equipo de Desarrollo.

Cabe mencionar que dentro de las iteraciones de construcción se llevan a cabo actividades relacionadas con Requerimientos, Diseño, Testing, etc., ya que se trata de un proceso iterativo e incremental es que se va llevando a cabo tareas en paralelo y la aplicación va evolucionando hasta cumplir los casos de uso definidos.

Disciplinas dentro de las Fases

Para planificar un proyecto y permitir tener visibilidad sobre el progreso y el grado de avance del mismo, AgEnD propone dividir el proceso en disciplinas que conforman agrupaciones de actividades en las cuales se produce un conjunto particular de artefactos. Las disciplinas cubiertas son las que se muestran en la Figura 014 y se irán realizando dentro de cada fase con diversos grados de paralelismo entre ellas.

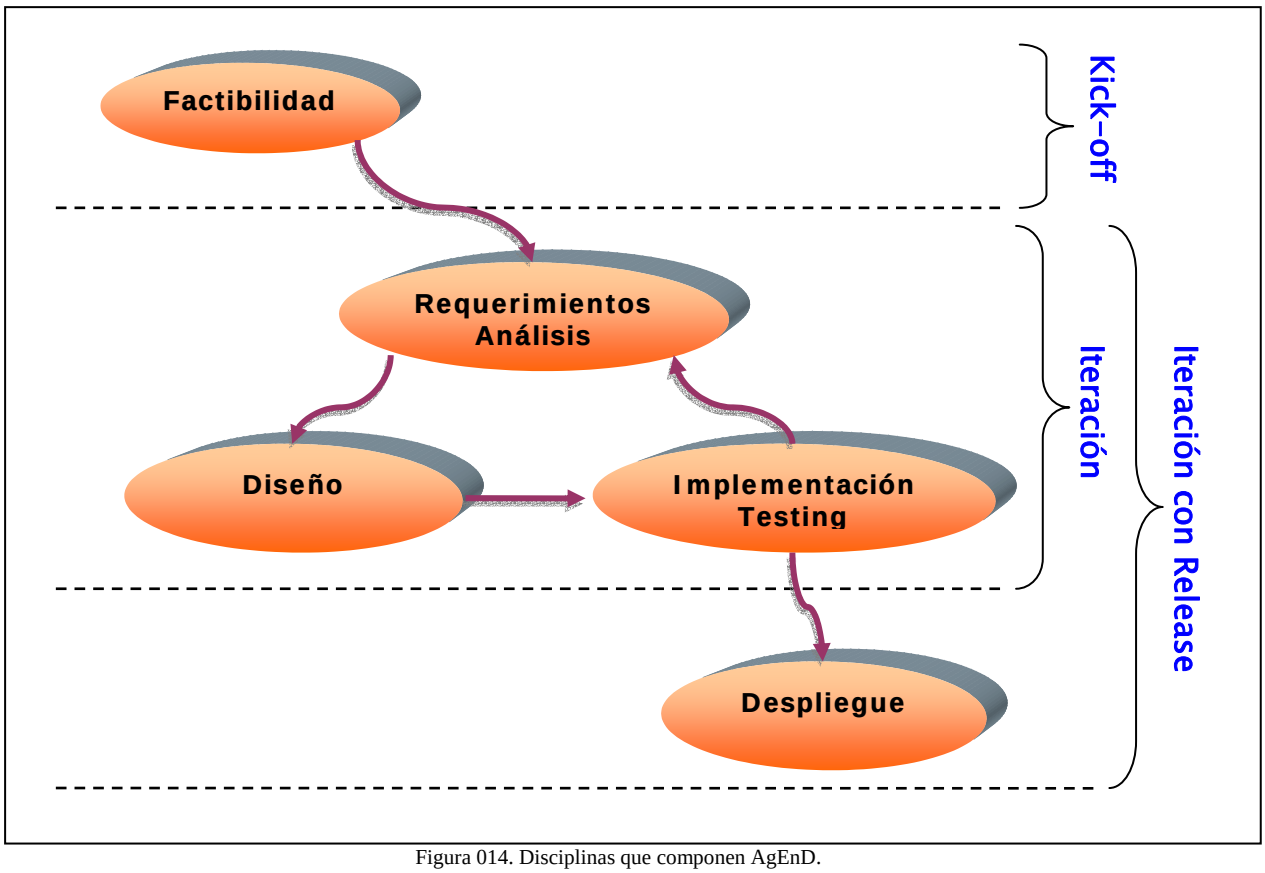

Figura 014. Disciplinas que componen AgEnD.

Factibilidad

Durante la disciplina de *Factibilidad* se produce el primer contacto entre el equipo de desarrollo y el cliente. Esta disciplina permite al equipo de desarrollo adquirir todo el conocimiento posible acerca del dominio, las necesidades de los usuarios, los objetivos y expectativas que debe cumplir el software que está siendo construido. Mediante las actividades incluidas en esta disciplina, seremos capaces de responder las siguientes preguntas: ¿Es conveniente construir la aplicación? ¿Es conveniente para el cliente el desarrollar un sistema con todas las complejidades inherentes para satisfacer las necesidades del usuario? ¿Cuáles son las posibles soluciones que se pueden plantear?

A medida que se ejecuta esta disciplina, se va realizando el Modelado del Dominio tendiente a entender las operaciones que forman parte del dominio del problema, el dominio manejado por el cliente. En forma conjunta, los analistas comenzarán a entender el problema planteado comenzando a bosquejar cuáles son los features que tendrá el sistema y el arquitecto comenzará a realizar modelos y diagramas

planteando una arquitectura preliminar que pueda servir de solución al problema en cuestión.

Durante esta disciplina el énfasis está puesto en la exploración tanto del problema como de la solución. Es esencial el aprendizaje del dominio del cliente, de cómo el sistema puede solucionar los problemas que se presentan, de cómo esta solución puede ser realizada con las herramientas disponibles, de las restricciones impuestas sobre el sistema.

Como se observa en la Figura 015, el entendimiento del negocio permite al equipo de desarrollo conocer explorar soluciones adecuadas a las necesidades de los usuarios; este conocimiento puede ser plasmado en un documento de Modelo del Dominio que debe ser construido conjuntamente con los usuarios y validados por estos.

Mientras se va desarrollando el Modelo del Dominio, se van explorando las distintas posibilidades en relación al dominio de la solución. La solución técnica propuesta deberá ajustarse a los estándares que posea el cliente, y el equipo de desarrollo deberá asegurarse de la factibilidad de su implementación. Para este punto se puede tener un documento de Descripción de Arquitectura que detalle las decisiones técnicas tomadas a lo largo de la *Factibilidad*.

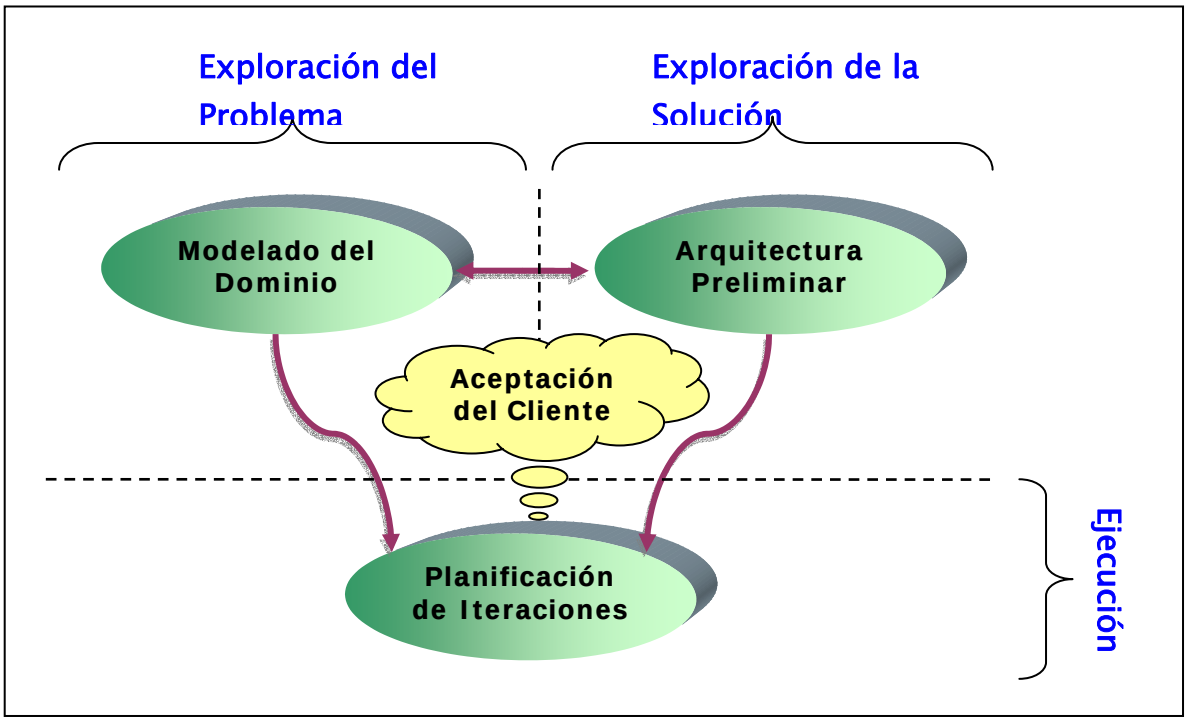

Figura 015. Factibilidad en detalle.

Como se observa en el detalle, una vez que se finaliza la exploración y se han creado los documentos que comuniquen el resultado de la misma al cliente, se deberá tener concurrencia de este y de los demás stakeholders en relación al compromiso para que el proyecto sea continuado o cancelado. Es de gran importancia el contar con el soporte ejecutivo necesario para proseguir esta disciplina, ya que según lo reportado por [Standish, 1994] la segunda razón por la cual un proyecto puede alcanzar el éxito es *Soporte de la Gerencia Ejecutiva*. El momento para garantizar este soporte es posteriormente a la exploración de los dominios ya mencionados, justo antes de la planificación de las iteraciones en que será construido el sistema.

La *Factibilidad* finaliza con la planificación del contenido funcional que será desarrollado en cada una de las iteraciones en que se construirá la aplicación. En dicho plan, se determinarán cuales serán las iteraciones con release, las cuales servirán como hitos del progreso del proyecto tanto para el cliente como para el equipo de desarrollo.

Dicha planificación reviste un carácter preliminar ya que los requerimientos pueden presentar cambios que alteren el contenido de las funcionalidades a ser implementadas en cada iteración. Está planificación contendrá los elementos más importantes y será plasmada en un artefacto denominado Plan de Proyecto, el cual será mostrado en detalle en la sección de entregables.

Requerimientos-Análisis

Una vez completadas las actividades dentro de la disciplina de *Factibilidad*, comienza el desarrollo netamente iterativo. Las iteraciones se suceden mediante la realización de actividades relacionadas con las subsiguientes disciplinas. A continuación, explicaremos en detalle la primera disciplina con que comienzan las iteraciones de AgEnD.

En la disciplina de *Requerimientos-Análisis* se realizan actividades tendientes a capturar las necesidades de los stakeholders, transformar las mismas en features de alto nivel y especificarlas posteriormente en casos de uso. Como se observa en la Figura 016 se deberá especificar el espectro funcional el cual servirá para que los desarrolladores diseñen e implementen los casos de uso y también el espectro no funcional que

posteriormente utilizará el arquitecto para construir la arquitectura de la aplicación y el prototipo.

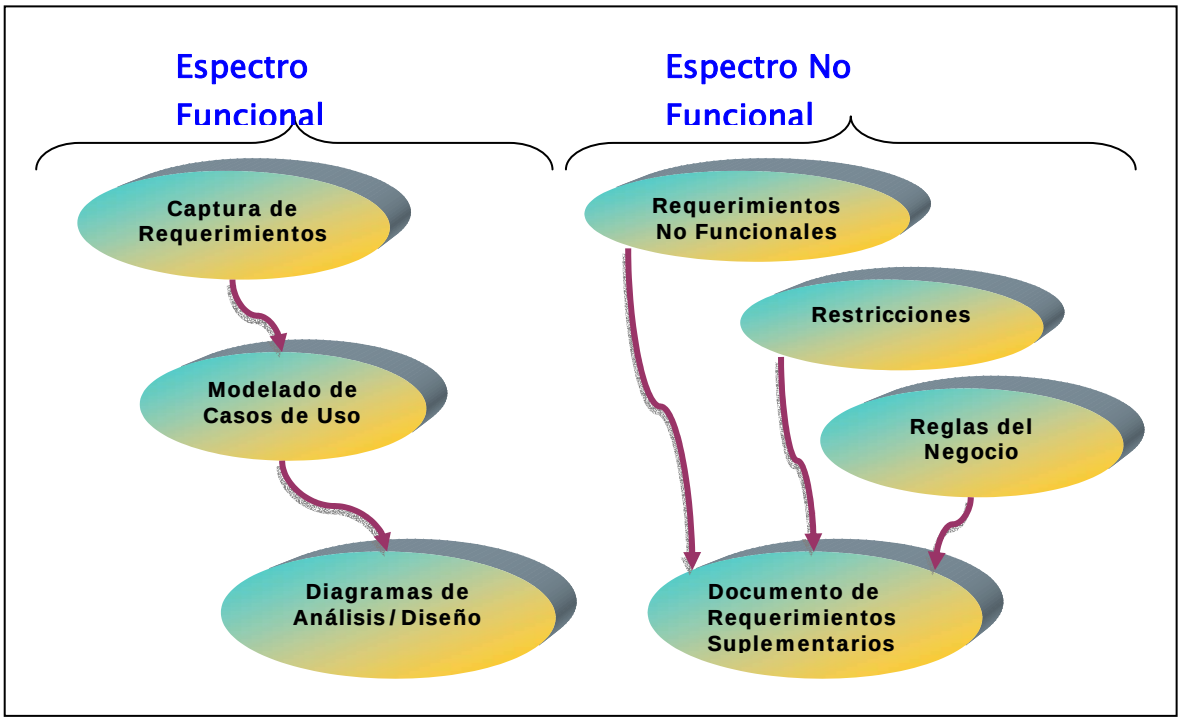

Figura 016. Requerimientos-Análisis en detalle.

Dadas las características de AgEnD tendientes a seguir un proceso ágil que puede administrar requerimientos altamente volátiles se sugiere utilizar el patrón denominado *Orientación al Cliente*, el cual consiste en maximizar la comunicación directa con las personas que terminarán utilizando el sistema para asegurarnos de estar construyendo lo que estas necesitan. En otras palabras el proceso será más efectivo si se tiene a un Cliente al menos trabajando como parte integral del equipo de desarrollo [Beck, 2000]. Su principal función será la de instruir y transferir su conocimiento al resto del equipo a medida se transcurre el proyecto. Muchas veces esto resulta impráctico en cuyo caso se deberá asegurar el involucramiento de los usuarios y una frecuente validación por los mismos de los artefactos construidos en cada fase.

Para llevar a cabo esta transferencia de conocimiento desde los usuarios y stakeholders que forman parte del proyecto a los analistas del equipo de desarrollo, se utilizan diversas técnicas las cuales han sido tratadas extensivamente en la bibliografía del tema; se puede ver en [Leffingwell, 2001][Kulak, 2003] una descripción de las más relevantes. De estas técnicas que se observan, AgEnD recomienda utilizar aquellas que

no demanden gran cantidad de recursos o una significativa infraestructura. Una de las técnicas más utilizadas en los procesos ágiles son las entrevistas, las cuales tendrán un carácter más informal ya que se podrán dar en cualquier momento que el analista deba aclarar cuestiones respecto a los requerimientos, detallar aspectos de un caso de uso, etc. En caso que se necesite hacer participar a un número importante de stakeholders en el relevamiento, se recomiendan sesiones de brainstorming dirigidas por el Líder de Proyecto a las cuales asisten los analistas documentando todo las decisiones tomadas.

Posteriormente a la Captura de Requerimientos se realiza la primera aproximación de los mismos en una forma estructurada que consiste en los casos de uso, propuestos por Ivar Jacobson a principios de los ’90. Los mismos se han transformado en un estándar en la industria de la informática y han trascendido más allá de la comunidad de objetos de donde surgieron para ser utilizados ampliamente en proyectos de todo tipo en el marco de IS/IT. La principal ventaja de los mismos es que son escritos en el lenguaje del cliente y no demandan tener conocimientos técnicos para ser entendidos. Son esenciales en los procesos ágiles en que se intenta que exista comunicación constante y fluida entre los miembros del equipo y el Cliente.

Para escribir los casos de uso se recomienda utilizar un formato predefinido como el que se encuentra en el Anexo a esta tesis. Esto permitirá que los analistas del equipo de desarrollo sincronicen el nivel de ambigüedad de la narración, los atributos a ser escritos y las convenciones que se utilizarán. Más material respecto al nivel de detalle con el que se puede realizar un caso de uso y todo aquello relacionado con su construcción ha sido estudiado por [Cockburn, 2000a].

Diseño

La disciplina de *Diseño* consiste en plasmar el problema planteado (que se halla en forma de casos de uso o requerimientos contenidos implícitamente por Analistas o Clientes) en el dominio de la solución. Asimismo, también se realiza la definición de la arquitectura siendo validada mediante algún prototipo (conocidos como *Proof Of Concept*).

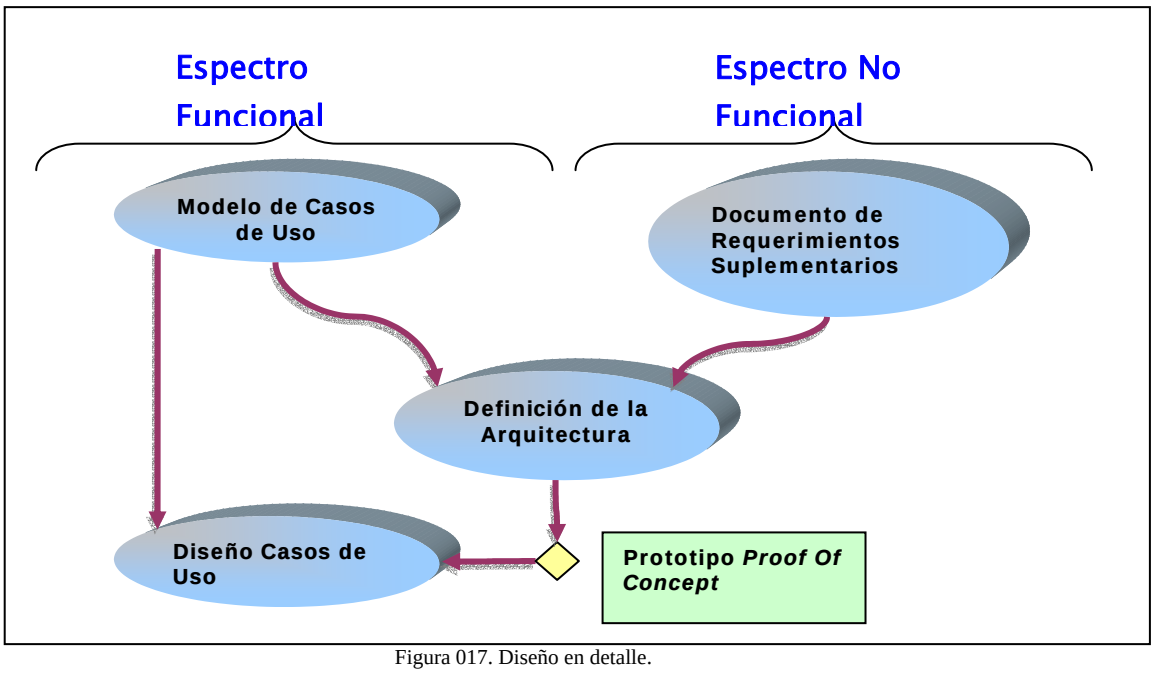

Figura 017. Diseño en detalle.

Durante las primeras iteraciones cabe destacar la necesidad de diseñar la infraestructura sobre la cual será construido el software. Para esto el arquitecto tomará del conjunto de casos de uso relevados hasta el momento aquellos que considere más importantes desde un punto de vista técnico y también analizará los requerimientos suplementarios que deberán ser satisfechos por la arquitectura propuesta. Con esto construirá un modelo de arquitectura documentado que especificará los elementos a partir del cual será construido el sistema, las interacciones entre dichos elementos, los patrones que se usaron para su composición, y las restricciones sobre dichos patrones. Finalmente, para probar la factibilidad de la arquitectura el Arquitecto construirá un prototipo ejecutable utilizado para verificar la conformancia a los requerimientos de la aplicación.

Dentro de AgEnD, el Diseño es realizado tomando como input los casos de uso escritos en la disciplina anterior y detallando los mismos en un esquema de clases, atributos y métodos del lenguaje elegido. Los miembros del equipo adaptan su modelo mental para cerrar la brecha que existe del dominio del problema, el *qué*, al dominio de la solución, el *cómo*. Este proceso es plasmado mediante la generación de Diagramas de Secuencia para aquellas interacciones significativas desde un punto de vista técnico las cuales podrán ser anexadas al Documento de Arquitectura. El nivel de ceremonia estará acorde a la criticidad del sistema que se desarrolla. En casos minimalistas y donde se desee tener un mínimo overhead metodológico el diseño podrá ser realizado en papel o

en un pizarrón, en casos de mayor ceremonia se utilizarán herramientas de modelado visual que ayuden en este proceso.

Una función esencial es la de asignación de funcionalidad en que el Líder de Proyecto o el Arquitecto se encargarán de nombrar a un responsable por cada unidad de funcionalidad especificada. Puede tratarse de un caso de uso, de un grupo de clases, de un paquete. Lo importante es que exista un responsable para cada artefacto que mantenga actualizada la trazabilidad requerida por AgEnD pudiendo realizar cualquier tipo de modificación sobre el mismo y manteniendo un adecuado control de cambios.

Una vez finalizado el proceso de diseño detallado de las clases a ser construidas durante la iteración, se procede a la siguiente disciplina que es la de *Implementación-Testing*.

Implementación - Testing

En la disciplina de *Implementación-Testing* se lleva a cabo la producción del software. El equipo de desarrollo se encarga de implementar la funcionalidad especificada en los casos de uso mediante el lenguaje de programación elegido.

Dado el énfasis que AgEnD propone en reducir la brecha comunicacional entre las personas que forman el equipo, esto mismo repercute en la preferencia de elegir para proyectos de esta envergadura aquellos lenguajes de programación que minimicen la brecha entre la realidad y la realización de la misma en un lenguaje. Idealmente, AgEnD fomenta el uso de frameworks o tecnologías bajo el paradigma de orientación a objetos, como *Java, MS. NET o Smalltalk*, los cuales permiten el máximo de eficiencia por línea de código para aplicaciones de IS/IT. Según se observa en la Tabla de Lenguajes de Programación incluida dentro de la sección de Anexos de esta tesis y relevada por [Jones, 1997], los lenguajes de este tipo permiten una menor cantidad de líneas de código por puntos de función, maximizando así la productividad del desarrollador.

En paralelo a la construcción de los componentes se realiza la revisión de los mismos mediante el testing, el cual tiene una gran relevancia en AgEnD. Siendo está una de las maneras en que se llevan a cabo los mecanismos de SQC (Software Quality

Control), la metodología define diversas pruebas que podrán ser realizadas en las iteraciones. Las mismas incluyen:

- Pruebas Unitarias:

  Definición: las Pruebas Unitarias son pruebas realizadas a nivel de las clases y métodos construidos para la aplicación. Estas pruebas son implementadas por los mismos programadores en el lenguaje de programación utilizando clases de prueba encargadas de verificar el comportamiento correcto de los métodos en una clase. Para ello es conveniente contar con algún framework de testing unitario que le facilite al equipo de desarrollo la creación, mantenimiento y ejecución de una suite de pruebas automatizadas. Ejemplos de esto pueden leerse en [Beck, 2002][Crispin, 2002][JUnit, 2001].

  Alcance: las mismas son utilizadas para controlar la calidad del código construido y son ideales para testing de integración y de regresión.

- Pruebas Funcionales:

  Definición: las Pruebas Funcionales verifican el flujo de interacción de la funcionalidad definida en los casos de uso. Esta funcionalidad – que está escrita en forma narrativa en los casos de uso y es plasmada mediante la interacción de objetos enviándose mensajes entre sí en los diagramas de secuencia – debe ser especificada en pruebas automatizadas que serán creadas por el Usuario en conjunción con la ayuda técnica del Tester y serán ejecutadas por el Tester durante cada iteración. Cabe remarcar que debe ser el Usuario el que define el contenido de las mismas ya que de esta forma proveen el feedback que permite al equipo de desarrollo continuar las iteraciones asegurándose de la calidad del producto construido.

  Alcance: las mismas son utilizadas durante todo el desarrollo y su alcance está definido por el conjunto completo de requerimientos funcionales definidos en los casos de uso.

- Pruebas de Aceptación:

  Definición: el objetivo de las Pruebas de Aceptación es la verificación que el software construido en las iteraciones cumple con las funcionalidades solicitadas por el usuario y está listo para ser utilizado en el ambiente del Cliente. Estas pruebas revisten gran relevancia porque implican que el Cliente da conformidad respecto al sistema desarrollado y lo acepta

  Alcance: las mismas son realizadas durante las *Iteraciones con Release* en las que se lleva a cabo la transición de la aplicación al ambiente de producción en el Cliente.

Despliegue

La disciplina de *Despliegue* permite que el sistema construido sea transferido al ambiente de producción para ser utilizado por el usuario. Esto no significa que el software debe estar implementado en su totalidad, cubriendo el 100% de los aspectos funcionales y no funcionales sino que en una iteración determinada se desee liberar una versión con un porcentaje de los casos de uso implementados. Dado el esquema de desarrollo iterativo e incremental, durante las iteraciones ya mencionadas se va agregando funcionalidad y se hace "crecer" al sistema hasta que cumpla con los requerimientos que se especificaron al principio, sumados a los cambios surgidos a lo largo del desarrollo.

Durante esta disciplina, se deben realizar actividades tendientes a hacer una entrega completa con una funcionalidad previamente determinada la cual será desplegada en producción. Las actividades incluidas dentro de esta disciplina son:

- Generar Manuales del Usuario, de Operación, etc.
- Realizar un empaquetamiento de componentes junto con scripts de instalación que el Cliente utilice para instalar la aplicación en su entorno.
- Ejecutar el conjunto completo de pruebas funcionales creadas durante el desarrollo, hasta obtener la aceptación del Cliente.

- En caso de ser necesario, definir las actividades de migración al nuevo sistema.
- Capacitar a los Usuarios que utilizarán el nuevo sistema.
- Generar la Nota de Entrega con la totalidad de los artefactos que se le entregan al Cliente (con su correspondiente versionado) al final de la disciplina.

El Líder de Proyecto deberá encargarse de lograr el consenso de todos los involucrados en una fecha de entrega del sistema. Para esto coordinará las fechas de forma que el Cliente tenga la oportunidad de verificar el correcto funcionamiento del sistema, mediante las pruebas de aceptación que se han generado hasta el momento. Es conveniente, que se planifique una reunión formal entre las dos partes (Cliente y Equipo de Desarrollo) para que todos estén en conocimiento de los resultados obtenidos, se puedan realizar comentarios respecto a las cuestiones que se deberían mejorar en el proceso para las siguientes iteraciones. En otras palabras, esta reunión serviría como una revisión Post-Mortem en la que se evaluarían todas las iteraciones de desarrollo que precedieron a la actual, y cuáles fueron las actividades que funcionaron exitosamente y cuáles fueron las dificultades encontradas en el camino.

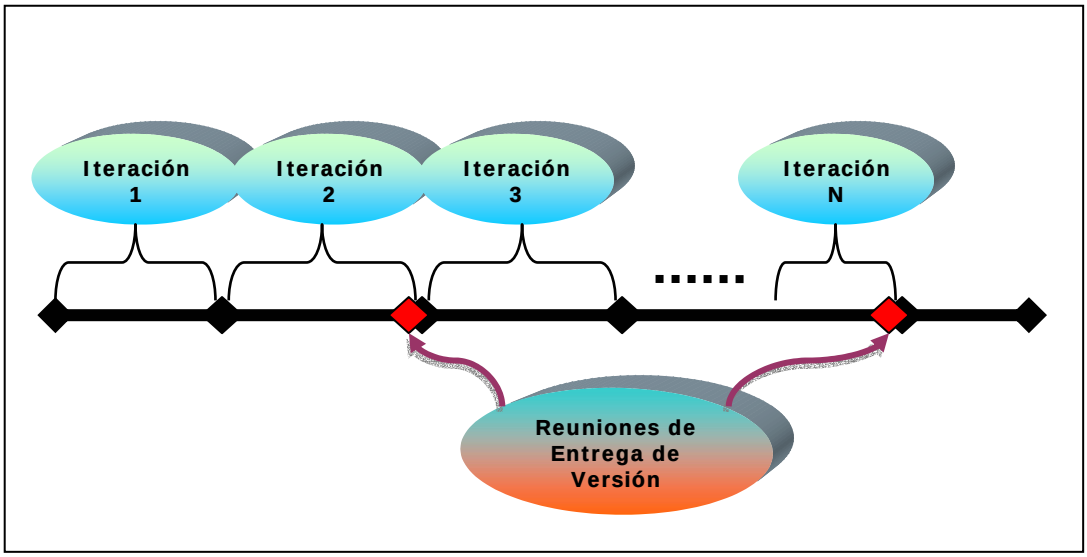

Figura 018. Esquema cronológico de reuniones de entrega de una versión del sistema.

Como se observa en la Figura 018, las Reuniones de Entrega de Versión se realizan al final de cada iteración de release. En este caso, la Iteración 2 va a ser la primera entrega de funcionalidad en el proyecto. Toda la funcionalidad desarrollada a lo largo de las dos primeras iteraciones será liberada al Cliente al fin de la Iteración 2. Por ello, se planifica la Reunión de Entrega de Versión al final de la misma. Este caso es equivalente a lo que ocurre en la Iteración N.

Disciplinas de Soporte

Las disciplinas de soporte ocurren a lo largo de todo el ciclo de vida del proyecto y dan soporte a las actividades relacionadas con la construcción del software. Las mismas son tan importantes como las disciplinas ya mencionadas pero la diferencia es que sirven propósitos organizacionales no directamente relacionados con la producción de sistemas. Entre estas nuevas disciplinas mencionaremos cuatro como muestra la Figura 019.

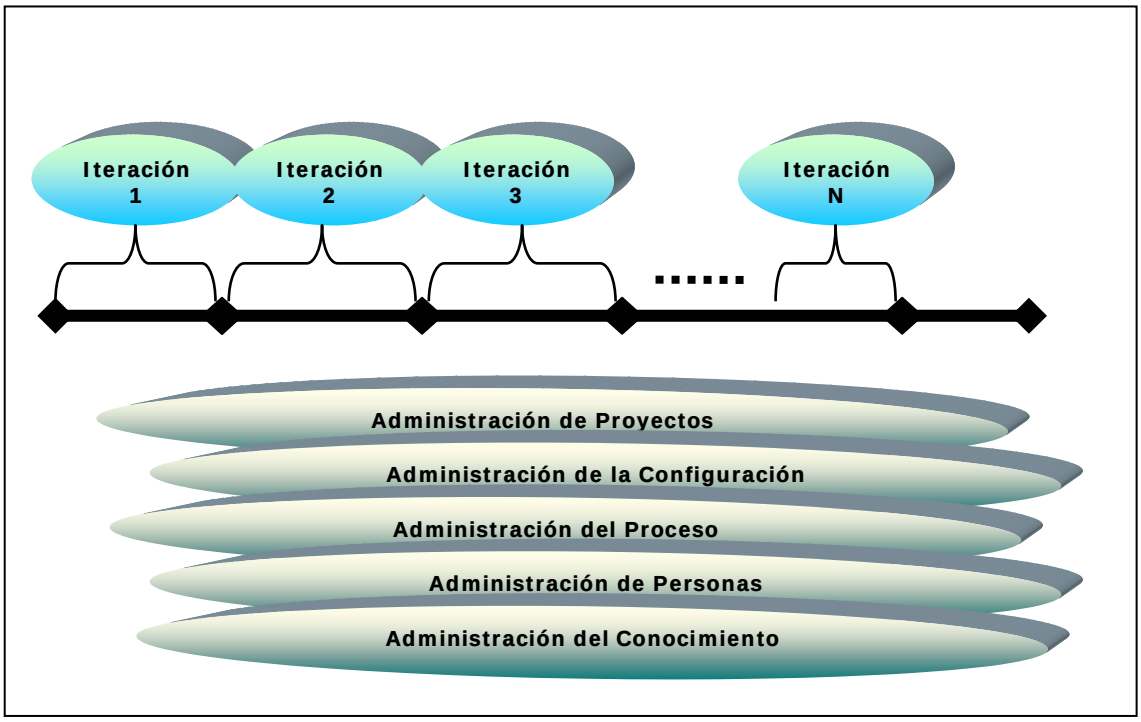

Figura 019. Disciplinas de Soporte de AgEnD.

Administración de Proyecto

La disciplina de *Administración de Proyecto* engloba todas las actividades relacionadas con la gestión del emprendimiento. Dentro de AgEnD esto refiere a las tareas que el Líder de Proyecto lleva a cabo para garantizar un ambiente saludable de trabajo, una medición del grado de avance del proyecto, un continuo monitoreo y mitigación de riesgos. Estas actividades permiten que el proyecto llegue a su éxito, entregando la aplicación en tiempo, forma y dentro de los costos presupuestados.

Dentro de AgEnD el rol del Líder de Proyecto es bastante distinto a las metodologías tradicionales basadas en la idea del control jerárquico. Bajo este paradigma el Líder era quien repartía el trabajo a las personas y quien estimaba el tiempo que llevarían todas las actividades mediante la realización de un WBS detallado y continuamente monitoreaba el trabajo en relación a sus estimaciones para aplicar correcciones. Bajo AgEnD el Líder no es más que la persona que habilita al equipo de desarrollo para que pueda trabajar cómodamente, sin obstáculos, resolviendo las necesidades que esta tenga. Tiene el carácter y las responsabilidades de un facilitador. Asimismo, es la cara visible del equipo hacia los stakeholders del proyecto – salvando los stakeholders claves que mencionaremos posteriormente. Deberá reportar y conocer el status del grupo y cómo el desarrollo se va dando en relación a la planificación establecida.

AgEnD recomienda que se establezca una planificación inicial del proyecto que permita tener objetivos claros respecto a los grandes hitos para obtener feedback del cliente respecto al avance realizado. Se armará un plan de proyecto con las fases y las iteraciones durante las que se irá construyendo el sistema. Dadas las características de este proceso ágil, no se debería realizar un plan ultra detallado con un WBS que presente todas las tareas a realizarse. Simplemente, se fraccionará la funcionalidad relevada hasta el momento en iteraciones en las cuales se irán liberando versiones al cliente para obtener el tan valioso feedback (ver en esta tesis la sección de *Estimaciones Ágiles* y la de *Orientado a los Productos vs. Orientado a las Tareas*).

Es importante destacar que el plan es armado consultando al equipo de desarrollo; la única tarea del Líder de Proyecto es armarlo en algún formato presentable al cliente, como por ejemplo un diagrama Gantt o una planilla Excel, dependiendo del nivel de formalidad requerido. El equipo de desarrollo será el que estimará la duración de las tareas a este nivel, entendiendo que la estimación será de una precisión muy

limitada, ya que no se cuenta con mucha información de la funcionalidad a desarrollar. En cada iteración el plan será actualizado para reflejar la realidad.

Una de las prácticas importantes es la de llevar a cabo una reunión diaria de no más de 15 minutos emulando a la *Scrum Daily Meeting* recomendada por Scrum. La misma refuerza el patrón de *Máxima Comunicación* (ver más adelante en esta tesis) que describe la necesidad de que el flujo de interacción sea frecuente; en este caso para que el Líder de Proyecto pueda medir el avance realizado diariamente, resolviendo cualquier problemática del equipo que este a su alcance y eliminando obstáculos. El cliente podrá participar de la misma pero sin poder esgrimir comentarios, sólo como un oyente pasivo.

Administración de la Configuración

La disciplina de *Administración de la Configuración* contiene las actividades relacionadas con la administración del repositorio que almacena los ítems de configuración generados en un proyecto. Esta es un área de conocimiento dentro de la Ingeniería de Software en la que mayores avances se han realizado y que ha estado presente en los proyectos de software desde aproximadamente 1980. Actualmente la mayor parte de los desarrollos incluyen alguna herramienta de manejo de SCM que automatiza todas las operaciones sobre el versionado.

AgEnD recomienda colocar todos los artefactos generados o consumidos en un proyecto bajo control de configuración. Esto permitirá que en cualquier momento se pueda recuperar una versión determinada de un ítem o ítems de configuración para recrear la cronología del proyecto o volver cambios atrás, o llevar a cabos registros de trazabilidad y/o auditorias entre otras cosas.

En particular, el uso de una herramienta de SCM le da la posibilidad al equipo de desarrollo de tener prácticas eficientes al momento de realizar las actividades en un proyecto. Entre estas prácticas cotidianas presentadas en cualquier desarrollo podemos mencionar:

- Los desarrolladores pueden trabajar en un mismo proyecto, compartiendo todos los ítems de configuración.

- Los desarrolladores pueden compartir el esfuerzo en el desarrollo de cualquier artefacto, sea una clase, un documento, etc.
- Los desarrolladores pueden acceder la versión estable actual del sistema para verificar si el código que generaron seguirá funcionando cuando sea integrado.
- Los desarrolladores pueden volver a una versión anterior del sistema para hacer pruebas contra esta.
- Los desarrolladores pueden trabajar en distintas ramas en paralelo sobre los artefactos del sistema, permitiendo tener una rama estable, otra para experimentar, y así sucesivamente.

Es importante que las herramientas sean usadas consistentemente y que todo el equipo tenga el conocimiento para trabajar con ellas. Esto podrá requerir capacitación a las personas que lo utilizarán.

Dentro del marco ágil de AgEnD la práctica utilizada dentro de esta disciplina deriva de una práctica propuesta por XP denominada *Collective Code Ownership*. Está práctica recomienda que no existan "dueños" de los ítems de configuración subidos al repositorio sino que cada persona podrá interactuar con cualquier ítem, pudiendo modificarlo y subir una nueva versión. En cierto sentido el equipo de desarrollo es responsable por todo el sistema por lo tanto todos los miembros de este son "dueños" de todos los ítems. Esto es posible gracias a la fluida comunicación existente entre las personas que fomenta AgEnD y al grado de responsabilidad que tienen las personas.

Dentro de esta disciplina incluiremos aquellas actividades relacionadas con la Administración de Cambio. Es decir, un proceso formal que nos permita controlar la forma en que los cambios son implementados durante el desarrollo. Dadas las características de AgEnD, los cambios son tomados como una posibilidad de tener un mejor entendimiento de la aplicación a construir por lo cual serán priorizados por el Cliente de acuerdo a su valor del negocio e implementados según corresponda en una iteración determinada. Dependiendo del costo asociado a los mismos el Cliente puede decidir descartarlos. Es de suma utilidad disponer de una herramienta que automatice el

seguimiento de los cambios a lo largo de su ciclo de vida, notificando al equipo de desarrollo de los cambios de estado de cada uno a lo largo del proyecto.

Administración del Proceso

La disciplina de *Administración del Proceso* permite que AgEnD sea adaptado a las necesidades de las personas y del proyecto en cuestión. Se decidió incluirla como una disciplina separada[5] dado que reviste gran importancia dentro de un proceso ágil; en la sección titulada *Enfoque Sistémico* de esta tesis se procede a un análisis sobre los cambios inherentes a una implementación metodológica.

Una vez que la organización ya posee una metodología como AgEnD estandarizada en sus proyectos y conocida por todas las personas, la misma deberá ser adaptada de acuerdo al contexto de cada proyecto en que se utiliza. Es decir, habrá distintos proyectos con características dispares en cuanto a tamaño del equipo, grado de criticidad del sistema, stakeholders involucrados. Todos estos factores darán como resultado una necesidad metodológica única, con ciertos roles, ciertos artefactos, ciertas actividades, etc. Consecuentemente AgEnD dispone de una disciplina que permite adaptarla a las necesidades de la instancia de proceso en que se esté.

Será clave en esta disciplina la figura del Coordinador quien empezará con un análisis del proyecto, tomando la planificación original, la funcionalidad, el equipo de trabajo, el perfil del cliente, y con todo esto definirá como se empleará AgEnD en dicho proyecto. El Coordinador deberá tener un amplio conocimiento del espectro metodológico existente para poder extraer aquellos elementos de AgEnD que considere que no apliquen y para poder agregar elementos que no estén contemplados y que formen parte de otros procesos. En este sentido se puede recomendar una profunda capacitación en el RUP que muchas veces cumple la función de un metaproceso sirviendo como base de conocimiento para los ingenieros de software. También conviene estar informado sobre las novedades de la Alianza Ágil (http://www.agilealliance.com) que está compuesta por los principales creadores de metodologías ágiles mencionadas en la introducción de esta tesis.

[5] El Proceso Unificado de Rational [RUP, 2002] incluye a las actividades de customización de proceso dentro de la disciplina de Ambiente en la que también aparecen todas las consideraciones humanas. En AgEnD decidimos dividirla en dos partes: *Administración de Proceso* y *Administración de Personas.*

Cabe remarcar que la tarea de adaptación del proceso es llevada a cabo por el Coordinador en cooperación con el equipo de desarrollo. Esto resulta muy relevante ya que será este ultimo el usuario final de la metodología por lo cual el Coordinador consultará con los miembros más especializados para entender los mecanismos de desarrollo involucrados.

Una vez que la metodología haya sido configurada para un proyecto, se harán reuniones esporádicas (una buena práctica consiste en coordinarlas con el fin de cada iteración) en las que se evaluará la adecuación de AgEnD a las características del proyecto. Se analizarán los resultados positivos y negativos, pudiendo el Coordinador hacer cambios sobre la marcha para mejorar la forma de trabajo.

Finalmente, cuando el proyecto esté llegando a su fin, el Coordinador, con la ayuda del Administrador de Conocimiento, almacenará la información acerca de cómo el proceso fue usado, con el propósito de mantener un historial de adaptaciones realizadas por tipo de proyecto.

Administración de Personas

La última de las disciplinas recomendadas por AgEnD es la de *Administración de Personas* que agrupa todas aquellas actividades relacionadas con los aspectos humanos del desarrollo. Nuevamente la figura clave de esta disciplina será el Coordinador, con una importante cooperación del Líder de Proyecto.

Una de las primeras actividades dentro de esta disciplina es la de nivelar a los miembros del equipo de desarrollo. Es importante que cada uno tenga todos los conocimientos necesarios para encarar las actividades por las que es responsable. El Coordinador deberá llevar a cabo talleres de capacitación y mentoring hasta asegurarse de que las personas han aprendido. Esta capacitación refiere tanto a aspectos técnicos como a los relacionados con las demás disciplinas de AgEnD, incluyendo la descripción del proceso.

Otra de las actividades que será realizada en forma conjunta con el Líder de Proyecto es mantener la salud del equipo de desarrollo. En otras palabras velar porque la motivación del equipo sea alta, garantizar relaciones interpersonales positivas, minimizar las fricciones debidas a distintos tipos de personalidad, entender a cada

individuo para poder sacar lo mejor de sí mismo. Existe un solapamiento entre esta disciplina y la de Administración de Proyectos en este punto. Se destaca que el mismo no es tan importante desde un punto de vista conceptual pero la diferencia está marcada en que las actividades de esta disciplina tienen como objetivo el bienestar de las personas en el proyecto mientras que el management se relaciona con la consecución exitosa del proyecto.

La práctica recomendada para esta disciplina tiene que ver con el patrón de Máxima Comunicación que será descrito más delante. Es decir, si tenemos canales de comunicación suficientemente ricos las dificultades podrán ser resueltas, en la mayoría de los casos, utilizando el sentido común y la buena predisposición de la gente.

Administración del Conocimiento

Un proceso de desarrollo de software debe proveer un mecanismo que permita que el conocimiento generado en un proyecto sea mantenido dentro de la organización. En la industria de software moderna es vital poseer mecanismos para reutilizar el conocimiento que se va generando a medida que la organización va ejecutando distintos tipos de proyectos.

Para ello definimos los tres niveles de refinamiento hasta llegar al conocimiento; son: *datos, información* y *conocimiento*. Comenzando con el nivel inferior, los *datos* consisten en valores discretos y objetivos acerca de eventos, pero sin ningún contenido acerca de su importancia o relevancia; a partir de éstos podremos generar información. La *información* son datos organizados de forma de servir de utilidad para las personas que mediante el contexto la utilizan para encarar tareas o tomar decisiones. Finalmente, el conocimiento requiere entender la información y tiene que ver con la relación entre ítems de información, su clasificación y su meta-dato (información de la información). La *experiencia* es conocimiento aplicado.

Se propone la utilización de una *fábrica de conocimiento ágil* siguiendo la línea de teoría de Victor Basili [Basili, 1990]. La misma consiste en una estructura paralela al equipo de desarrollo, la cual tiene personas de dedicación exclusiva dedicadas a capturar, empaquetar, almacenar y distribuir a lo largo de la organización.

A partir de esta idea y su adecuación dentro de una metodología ágil en que tendremos pequeños grupos de trabajo, AgEnD define un rol específico que es el *Administrador del Conocimiento*. Éste podrá ser desempeñado en forma rotativa por diferentes personas dentro del equipo de trabajo. Su tarea consiste en documentar todo aquel conocimiento novedoso que haya sigo generado durante el proyecto para que pueda ser reutilizado por otras personas. Será necesario algún tipo de soporte de herramientas, el cual puede ser algo bastante sencillo como un *Swiki*[6] o algún producto de administración de contenidos.

Dado que nos manejamos en un contexto ágil, es imperativo que este patrón no genere una extrema burocracia dentro de la organización del proyecto. Por eso se definió un rol particular como ya fue mencionado el cual deberá entender la arquitectura del sistema y los aspectos funcionales más importantes para poder capturarlos en documentos que puedan ser recuperados y consumidos en el futuro. Esto es de suma importancia y tiene que ver con el aprendizaje organizacional sugerido por Tom DeMarco [DeMarco, 1999]. Asimismo se recomienda capturar todas aquellas experiencias humanas positivas y negativas que se tuvieron en el proyecto para poder agruparlas en formato de patrones y anti-patrones que puedan ser dispersados y aprendidos por todas las personas.

Un comentario muy importante en relación a la Administración del Conocimiento es la necesidad de invertir en la gente y en abocar por la motivación de los equipos de desarrollo. Si una organización no valora a las personas que la conforman, nunca podrá aprender. Si en una organización la gente trabaja poco tiempo y se va, habiendo mucho nivel de rotación, el aprendizaje será escaso. Puesto de otra forma por Tom DeMarco:

> **El aprendizaje está limitado por la habilidad de una organización de mantener a su gente.**

Artefactos

[6] Herramienta de administración de contenido, desarrollada por Ward Cunningham. Permite de manera sencilla la carga, edición, borrado y visualización de cualquier tipo de contenido a través de la web. Más información en http://www.swiki.org

Las metodologías ágiles en su mayoría tienden a minimizar la cantidad de artefactos generados en un proyecto. El énfasis está puesto en el código entregado y cualquier documento construido es visto como un overhead. Sin embargo, existen muchos artefactos necesarios para propósitos de comunicación, diseño, gestión sin los cuales un proyecto no llegaría al éxito. AgEnD enumera algunos artefactos que considera requisitos mínimos para tener madurez en el proceso de desarrollo. En la sección de Anexos de esta tesis se han colocado templates para la mayor parte de los artefactos aquí descritos.

- Documento de Visión:

  Definición: el Documento de Visión es realizado durante la fase de Concepción y sirve como contrato entre el Cliente y el Equipo de Desarrollo respecto a lo que se va a construir. En la Visión deberán identificarse los stakeholders del proyecto, la totalidad de features del sistema a ser construido y los requerimientos suplementarios que se detectaron en forma temprana. En forma optativa podrá incluirse un resumen de los casos de uso críticos del aplicativo.

  Responsable: la Visión es construida por un Analista Funcional en conjunto con el Líder de Proyecto. La misma debe ser mantenida para reflejar los cambios al alcance durante el proyecto.

- Plan de Proyecto:

  Definición: el Plan de Proyecto es un documento mantenido por el Líder de Proyecto con toda la información de gestión. El mismo suele incluir un Gantt con el cronograma y las tareas del proyecto. Dependiendo del nivel de formalidad se generará un WBS y/o algún otro tipo de plan a incluirse dentro de este (plan de testing, de comunicaciones, de administración de requerimientos, de administración de cambios, etc.). El Plan de Proyecto es creado en forma temprana durante la Concepción y actualizado durante las fases posteriores.

Responsable: el Líder de Proyecto es el responsable de este documento y deberá mantenerlo actualizado hasta la terminación del proyecto.

- Lista de Riesgos:

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

  Responsable: el Líder de Proyecto es el responsable y el principal consumidor de este documento y deberá mantenerlo actualizado hasta la terminación del proyecto.

- Modelo de Casos de Uso y Casos de Uso:

  Definición: los Casos de Uso se utilizarán para especificar los requerimientos funcionales de la aplicación. Los mismos servirán para guiar los demás artefactos y para la construcción de los componentes de la aplicación. Asimismo, AgEnD recomienda la generación del Modelo de Casos de uso para tener una visualización gráfica del universo funcional de la aplicación y poder comunicarla tanto internamente como al Cliente para su validación. Los Casos de Uso y el Modelo serán creados en las primeras fases del proyecto, aunque podrán también ser especificados en las iteraciones de construcción.

  Responsable: los Analistas son responsables de la creación y el mantenimiento de estos artefactos, los cuales serán consumidos por el resto del equipo de desarrollo.

- Documento de Especificación de Requerimientos de Software (SRS):

  Definición: el documento de SRS se utilizara para especificar los requerimientos funcionales de carácter más técnico de la aplicación.

También en este se incluirían los requerimientos no funcionales, las reglas de negocio, y las restricciones que se aplicarán a la solución. En este documento se pondrán todos los requerimientos que queden fuera de los casos de uso. La idea es tener algún tipo de constancia documental de todos aquellos requerimientos de carácter técnico que debe contemplar la solución.

Responsable: los Analistas son responsables de la creación y el mantenimiento de este artefacto, el cual será consumidos por el resto del equipo de desarrollo.

- Descripción de la Arquitectura:

  Definición: el documento de Descripción de la Arquitectura también conocido como SAD especifica los aspectos técnicos de la solución propuesta por el equipo de desarrollo. Sirve como medio de comunicación entre el Arquitecto y el equipo en relación a cómo deberán ser implementados los casos de uso de la aplicación.

  Responsable: el Arquitecto es el principal responsable por la realización de este artefacto, así como la creación de un prototipo ejecutable (Proof of Concept) que valide que la misma cumpla los requerimientos no funcionales y las cualidades sistémicas que hacen a los atributos de calidad no relacionados directamente con las necesidades del usuario.

- Casos de Prueba:

  Definición: los Casos de Prueba contienen la especificación de cómo será validado el sistema. Estos son realizados basándose en los casos de uso y planteando todos los escenarios posibles que éstos pueden contener. Dado que puede ser bastante burocrático realizar la totalidad de Casos de Prueba existentes en un sistema, se recomienda generar los más importantes y después anotar las variantes en un Caso de Prueba Estándar.

Responsable: el Tester es el principal responsable por la realización de este artefacto. El Tester será el encargado del diseño y la ejecución de los Casos de Prueba.

- Scripts de Despliegue:

  Definición: los Scripts de Despliegue son necesarios para la instalación del producto en un entorno determinado. El mismo automatiza las tareas de empaquetamiento, versionado, compilación, etc. Un ejemplo muy utilizado baja la plataforma J2SE es la herramienta ANT, que permite generar scripts de despliegue con bastantes funcionalidades.

  Responsable: algún Programador será responsable de la generación y mantenimiento de estos scripts.

- Planilla de Incidentes:

  Definición: la Planilla de Incidentes será utilizada para registrar y tener seguimiento de aquellos bugs, mejoras, tareas que el Tester o alguna persona de aseguramiento de la calidad necesite reportar. Son los denominados sistemas de Issue Tracking que pueden ser implementados manualmente o mediante alguna herramienta.

  Responsable: la Planilla de Incidentes será utilizada por todos los miembros del equipo. En particular será muy utilizada por los Testers quienes cargarán todos los bugs encontrados durante la ejecución de los Casos de Prueba los cuales serán leídos y corregidos por los Programadores.

- Repositorio del Proyecto:

  Definición: el Repositorio es la herramienta fundamental para cubrir la disciplina de Administración de la Configuración. En el repositorio se almacenan todas las versiones de todos los archivos y directorios del proyecto. AgEnD recomienda que todo artefacto generado durante un

proyecto sea puesto bajo control de versiones y no sólo el código fuente de la aplicación. Esto permitirá en cualquier momento poder comparar versiones, volver a recuperar versiones anteriores, y generar ramas de desarrollo paralelo.

Responsable: algún Programador o alguna persona de Infraestructura de la organización será responsable de la creación, mantenimiento y eliminación (en casos que se desee dar de baja el proyecto) del repositorio.

- Nota de Entrega:

  Definición: la Nota de Entrega sirve para especificar aquello que se le entrega al Cliente en un release determinado de la aplicación. En la misma se constatará el número de versión, los cambios de la versión anterior, los bugs conocidos y las mejoras efectuadas.

  Responsable: cualquier miembro del equipo de desarrollo que vaya a entregar algún artefacto al Cliente deberá crear la correspondiente Nota de Entrega.

Patrones de Desarrollo Recomendados

Los patrones surgieron del universo de la arquitectura del trabajo de Christopher Alexander durante los años 1977-1979. En su primer libro de esta serie [Alexander, 1977] el autor escribe: “Cada patrón describe un problema que ocurre una y otra vez en nuestro ambiente, y después describe la base de la solución a dicho problema de una forma que se puede utilizar dicha solución un millón de veces sin realizarla dos veces de la misma manera”. Posteriormente Alexander simplifico la definición teniendo en cuenta que un patrón plantea un problema y después propone una solución al mismo. También nos da un contexto para entender cuándo la solución debe ser aplicada. Los patrones de Alexander no tuvieron una influencia muy importante en la comunidad de arquitectura, pero sí tuvieron un gran impacto en la industria del software alrededor de 1995, cuando se editó el libro de *Gang Of Four* [Gamma, 1995]. Se tomarán estos conceptos para definir los patrones que AgEnD recomienda en el desarrollo de software.

Dichos patrones son los mencionados a continuación:

1. Máxima Comunicación
2. Comunicación Interna al Equipo
3. Participación Activa del Cliente
4. Estimaciones Ágiles
5. Enfoque en la Arquitectura
6. Integración Continua
7. Peopleware

Máxima Comunicación

Las metodologías ágiles solo tienen sentido si existe comunicación entre las personas. Esto no solo refiere a las personas del equipo de desarrollo, sino a todas las personas que se vean afectadas en distinta medida por la ejecución del proyecto; los denominados durante esta tesis como stakeholders.

Bajo el patrón de *Máxima Comunicación* se encuentra uno de los principios esenciales de AgEnD. Relacionado estrechamente con una de las actividades más comunes de los seres humanos: la comunicación – que representa las diversas formas de transmisión de datos/información/conocimiento/experiencia entre individuos.

El software se encuentra entre los elementos de mayor complejidad creados por el hombre. Dada esta complejidad inherente a su esencia, el software presenta una gran dificultad en su construcción caracterizada por la aleatoriedad de las posibilidades en el desarrollo de cualquier sistema y la incapacidad de concebir procesos repetibles – como ocurre en las demás ingenierías. Si bien se han logrado muchos avances en cuanto al manejo de dicha complejidad, la misma aún persiste y genera resultados no deseados como cronogramas excedidos, baja calidad o funcionalidad incompleta.

Existen dos tipos de comunicaciones que este proceso toma en cuenta para mejorar el desarrollo. Estas comunicaciones difieren en relación a las personas involucradas en las mismas. La primera es la comunicación interna, es decir la comunicación entre todos los integrantes del equipo de desarrollo – desde el líder del proyecto hasta el programador. La segunda comunicación a la que nos referiremos es la comunicación entre los integrantes del equipo de desarrollo y el cliente o usuario del sistema. Es decir, todas las comunicaciones que el equipo mantenga con los *stakeholders*. En la figura 020, tenemos una representación gráfica de los niveles presentados en este párrafo.

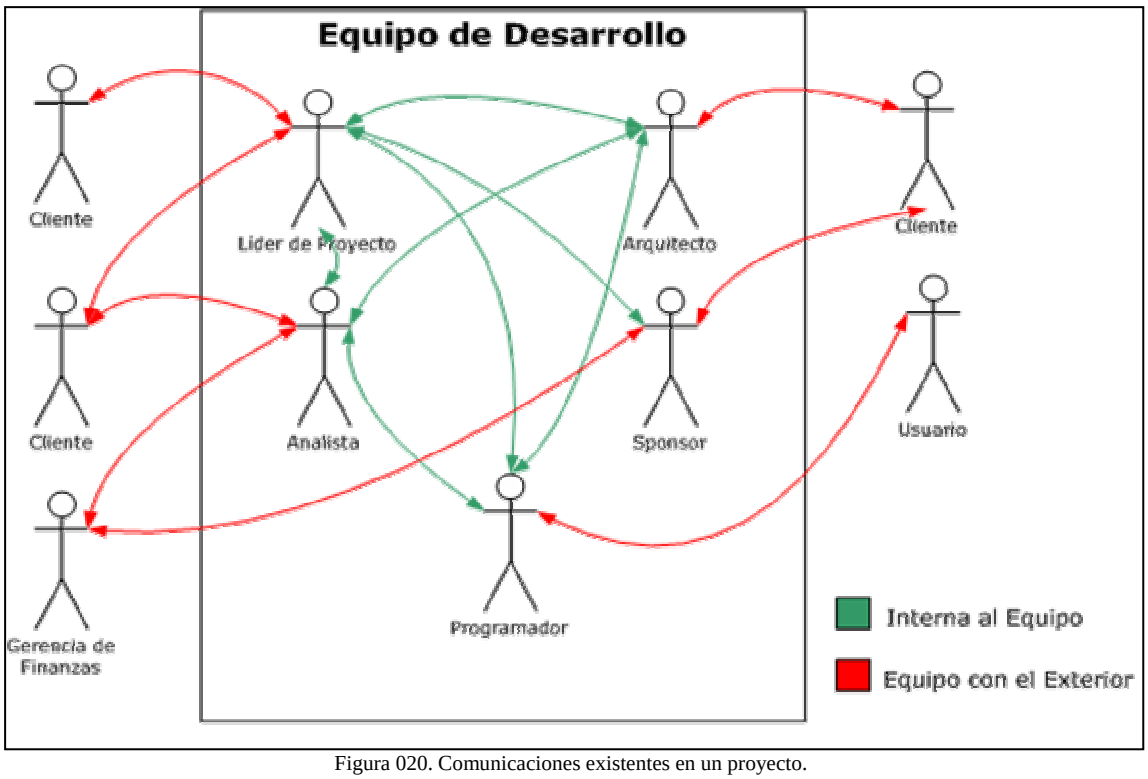

Figura 020. Comunicaciones existentes en un proyecto.

Esta división nos permite concentrarnos en las prácticas que podrán ser aplicadas en cada caso. Analizaremos las formas de atacar el desarrollo de manera de maximizar la comunicación entre las partes involucradas acorde al contexto en que se plantean.

Tomando los principios que guían a AgEnD, continuamente se priorizan las personas sobre el proceso. La metodología debe proveer mecanismos que fomenten las interacciones ayudando a mejorar la motivación y la productividad del equipo de desarrollo. De esta forma el flujo de información entre las mentes de los desarrolladores será maximizado permitiendo mayor progreso en el proyecto. Alistair Cockburn [Cockburn, 2001a] analizó este fenómeno en detalle evaluando el impacto de distintos modelos comunicacionales. La Figura 021 muestra la influencia en la efectividad de la comunicación de acuerdo al canal que se utiliza para establecer la misma.

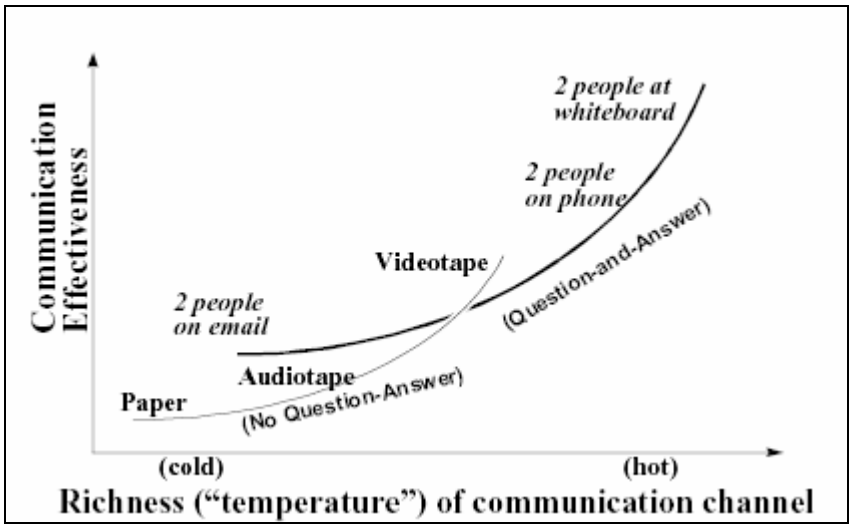

Figura 021. Efectividad de distintos modos de comunicación. Tomada de [Cockburn, 2001a].

Acorde a los resultados reportados en dicha investigación AgEnD recomienda diversas prácticas que estimulen canales de comunicación más abiertos tanto internamente al equipo como externamente con los stakeholders del proyecto.

Comunicación Interna

La primera recomendación que podemos tomar tiene que ver con el patrón de *Mínima Dispersión Física*. Según este patrón el equipo de desarrollo debe estar colocado físicamente en una única oficina que tenga una distribución centralizada al momento de trabajo con lugares privados para cuestiones extra laborales. Esta es la propuesta de XP la cual no hace más que obedecer al canal de comunicación más rico que es ver a las personas cara a cara para interactuar con ellas.

Asimismo otro patrón denominado *Interrupciones Mínimas* que refiere a la necesidad de liberar al equipo de desarrollo de interrupciones referidas a otros ámbitos. En principio, se recomienda que la oficina del equipo de desarrollo este separada mediante paredes del resto de la organización. Si esto no se logra, se tendrán flujos de información que no refieren al desarrollo y que empeorarán la calidad del ambiente diseñado para maximizar la comunicación – como planteaba el primer patrón mencionado. Tom DeMarco [DeMarco, 1987] indaga bastante sobre los efectos nocivos de mantener entornos de trabajo con mucho ruido e información innecesaria que atenta contra el trabajo de las personas.

También este patrón aplica a la necesidad de que las personas no estén continuamente interrumpiéndose entre sí, esto es evidente en los casos de una persona que juega un rol clave y que constantemente debe resolver dudas. La idea subyacente es la de tratar primero de evacuar las dudas mediante lectura de algún documento, si esto no funciona, se podrá pedir auxilio a los pares. Como último recurso se debería convocar a esta persona, en algunos casos se pueden juntar un número importante de dudas e ir anotándolas y preguntarlas todas de una única vez. Sin embargo, dado que la comunicación es fomentada AgEnD recomienda una interrupción antes que guardarse la pregunta y hacer alguna suposición al respecto.

Un tercer patrón que se recomienda es el que Cockburn describe como *Radiadores de Información*. Según este patrón se recomienda tener las paredes de la oficina totalmente libres dispuestas para que se cuelguen en ellas pizarrones, hojas grandes, diagramas, notas post-it, etc. Estos actúan como radiadores de información ya que permiten que cualquier miembro del equipo pueda tener acceso a esa información constantemente sin necesidad de andar indagando a las demás personas. Más aún, los radiadores sirven para mantener información crítica a la vista de todos la cual puede ser discutida, analizada, modificada y comunicada continuamente. Ejemplos de información que se puede plasmar en estos es:

- Diagramas de Arquitectura
- Avance del proyecto en función de los casos de uso implementados
- Resultado de la ejecución de los casos de prueba
- Diagrama de casos de uso

Una implementación alternativa es tener un sitio web que disponga continuamente de información del proyecto. Adicionalmente, puede servir como base de conocimientos del proyecto y de la organización.

Comunicación Externa

Tan importante como la comunicación interna resulta la comunicación con los stakeholders del proyecto. Primero debemos dividir a los stakeholders en dos franjas: stakeholders claves para el éxito del proyecto y stakeholders normales. Los stakeholders

claves serán aquellos que poseen conocimiento del dominio del sistema a construir. Son aquellos que aceptarán el sistema en cuestión si el mismo se adecúa a sus necesidades. Por otro lado, tenemos los stakeholders normales con los que el equipo interactúa ocasionalmente para algún propósito del proyecto. Estos últimos juegan un rol que puede tener importancia para la salud del proyecto pero que no impacta directamente sobre la funcionalidad de la aplicación construida.

AgEnD recomienda maximizar la comunicación con los stakeholders claves del proyecto. En el caso ideal se recomienda utilizar el patrón *On-Site Customer* derivado de XP que establece que durante todo el proyecto habrá un cliente real sentado en la misma oficina con el equipo de desarrollo, dispuesto a responder preguntas, resolver disputas, y priorizar las tareas llevadas a cabo. Cabe destacar que en este caso un "cliente real" refiere a una persona que utilizará el sistema una vez puesto en producción y que conozca la aplicación en su extensión para resolver distintas cuestiones que puedan darse.

Ocurre que en la mayoría de los casos los clientes tienen que hacer sus propios trabajos y no disponen de tiempo para estar continuamente formando parte del equipo de desarrollo. Dadas las realidades impuestas en la mayor parte de los proyectos, AgEnD propone disponer de un cliente designado con el rol de *Experto en el Dominio* mencionado anteriormente. El mismo debe estar dispuesto a contestar cualquier pregunta del equipo de desarrollo ya sea mediante algún canal de comunicación más frío de acuerdo a la Figura 021 (ej.: teléfono, videoconferencia, mail, etc.) en un tiempo razonable que no debería superar algunas horas. En caso de que el problema requiera de una comunicación más personal, el cliente deberá disponer de tiempo para visitar al equipo de desarrollo y discutir el tema personalmente o bien para recibir a algún miembro clave en sus propias oficinas.

El énfasis está puesto en mantener el proceso lo más ágil posible, y dado que la recomendación es construir casos de uso que manifiesten las interacciones a un nivel no muy detallado siempre existirán dudas que podrán ser mitigadas por los *Expertos en el Dominio*.

Respecto a los stakeholders normales, la recomendación es que los mismos sean identificados en forma temprana y sus necesidades sean relevadas para verificar que el sistema conforme a estos adecuadamente. Si bien no será necesario tener un canal

abierto tan flexible la comunicación con estos debe ser fluida y honesta ya que en algún punto el sistema tendrá influencia sobre ellos o viceversa.

Participación Activa del Cliente

AgEnD plantea la necesidad de contar con una incondicional participación del Cliente durante el proyecto de desarrollo. Dadas las características de las aplicaciones a las que el proceso apunta, los únicos capaces de definir qué debe contener el producto son los usuarios dado que serán sus necesidades las que se verán afectadas. En esto AgEnD es **inflexible**; no hay documento ni modelo ni herramienta que pueda sustituir el feedback provisto en forma directa mediante contacto cara a cara entre el analista y el cliente o usuario del sistema. No hay forma de alcanzar el éxito en un proyecto si no se cuenta con el compromiso y el soporte del sponsor. El apoyo del management garantiza que los obstáculos que se presenten – políticos, económicos, sociales – puedan ser removidos mediante su activa participación.

Las razones de la inflexibilidad en este punto son aquellas que han revelado cuales eran los factores que más contribuían al fracaso de los proyectos de software. Según estudios realizados por el Grupo Standish [Standish, 1994] la falta de involucramiento por parte de los usuarios y el poco apoyo del management están entre los factores antes mencionados. Si se analiza este hecho concluimos que es lógico establecer que si el usuario no participa en la construcción del software nunca este último va a satisfacer sus necesidades por ser el usuario el único que las conoce. En otras palabras,

> **El no involucrar al usuario durante el desarrollo de software nos permite construir un producto de software que sólo cumple las necesidades de aquellas personas que no van a utilizarlo.**

Para lograr esta activa participación del usuario deberemos poder contar con usuarios expertos en el dominio que puedan prestar todo el tiempo necesario para las necesidades del equipo de proyecto. Entre estas se incluye la posibilidad de que los analistas puedan delinear requerimientos con el grado de detalle necesario para que los

programadores puedan comenzar con el diseño. También los programadores deben poder consultar a dichos usuarios si se les plantean dudas respecto al dominio del problema o existen omisiones o inconsistencias en los requerimientos.

Los usuarios deberán tener amplio conocimiento del dominio para poder resolver cualquier duda del equipo respecto a las funcionalidades a desarrollar. También deberá priorizar continuamente el valor del negocio de lo que se construye en cada iteración para guiar funcionalmente al equipo.

Estimaciones Ágiles

Al estar Orientadas a los Productos, las metodologías ágiles permiten realizar estimaciones de menor granularidad en el día a día para obtener más precisión en la estimación de la entrega de artefactos en cada iteración, en detrimento de una visibilidad a largo plazo. Sin embargo, es importante realizar estimaciones realistas de la duración de los proyectos sobre todo al principio de los mismos.

La técnica que propone AgEnD es la estimación por *puntos de caso de uso (use case points).* La misma mide a los sistemas desde una perspectiva funcional y es independiente de la tecnología. Sin tener consideración del lenguaje, método de desarrollo, o plataforma de hardware utilizada, el número de puntos de casos de uso de un sistema permanecerá constante. La única variable es la cantidad de esfuerzo necesario para desarrollar un conjunto dado de puntos de caso de uso; por lo tanto el *Análisis por Puntos de Caso de uso* puede ser utilizado para determinar si una herramienta, un ambiente, un lenguaje es más productivo comparado con otros dentro de una organización o entre organizaciones. Este es uno de los puntos críticos y uno de los valores más importantes de dicho análisis. El método de puntos de caso de uso está cubierto y detallado en [Ribu, 2001].

Dentro de los proyecto de implementación bajo el alcance de AgEnD, el objetivo es poder hacer una estimación de la funcionalidad a ser entregada en cada etapa en función de la productividad del equipo de desarrollo. AgEnD requiere que la persona responsable de un producto, ya sea una clase, un caso de uso o un documento de administración de la configuración, realice la estimación. Esto va en contra de las metodologías tradicionales en las que es el líder de proyecto el encargado de realizar

todas las estimaciones del proyecto. En los procesos ágiles, al reconocer que las personas son lo más importante del desarrollo las mismas adquieren más responsabilidad en relación a los productos que entregan, debiendo estimar la duración del desarrollo en cada iteración. Esto no quiere decir que los Programadores estarán a cargo de hacer la estimación del proyecto completo - el Líder será el que lo realice consultando a todos los miembros del equipo, ya que este es responsable por la concreción del proyecto entero - pero si serán responsables por los productos que construyen durante las tareas que realizan todos los días.

AgEnD fomenta el concepto de estimación descentralizada. Cada persona estimará el fruto de sus tareas por la técnica que considere necesaria. Una consecuencia de esto es que las personas no sienten que se les ha impuesto un cronograma al que deben atenerse pudiendo ser reprendidas en caso de retrasos. El responsable de un producto es la persona más adecuada para medir su ritmo y estimar los tiempos necesarios. Al principio esto podrá resultar difícil para aquellas personas acostumbradas a tener fechas y tareas impuestas desde arriba, pero ahí es donde entra en juego el Coordinador para capacitar a los miembros del equipo y perfeccionarlos en las estimaciones futuras.

La estimación para una iteración dada será realizada durante la iteración previa, de forma de tener la planificación aprobada por el cliente previo a embarcar en las actividades correspondientes al equipo en el desarrollo. Esta estimación será efectuada por todo el equipo junto al cliente y en la misma será validada la funcionalidad a ser próximamente construida.

Enfoque en la Arquitectura

Una de las ideas propuestas por AgEnD consiste en una temprana definición de la arquitectura del sistema. La arquitectura no es un concepto que pueda ser explicado en un par de palabras. Consiste más bien en una suma de aspectos que están relacionados con el diseño de un sistema.

Ya sea que estemos construyendo un edificio, una prensa hidráulica o un software empaquetado, debemos tener algún elemento de alto nivel que refleje las decisiones que se van tomando en relación al diseño de la construcción. Haciendo un

paralelismo con el paradigma de objetos, considerando al sistema como un paquete a ser embebido en un ambiente específico, la arquitectura estaría dada por:

- la organización del mismo, definida por los paquetes o clases que contiene y las relaciones entre los mismos
- las interfaces que provee al mundo exterior
- los paquetes que requiere para su funcionamiento
- los servicios que provee, realizados a partir de métodos en clases, que deben dar valor a algún actor que interactúa con el sistema
- los patrones, estándares, frameworks utilizados para su construcción

Cabe destacar, que la arquitectura mantiene un cierto nivel de abstracción dentro de la disciplina de diseño. No contempla los detalles de más bajo nivel, como la implementación interna de las clases y métodos, ni analiza en detalle los protocolos a ser utilizados por los sistemas. Simplemente propone un conjunto de vistas [Kruchten, 2000] que enfatizan distintos aspectos del software. Los mismos tienen que ver con:

- organización lógica
- funcionalidad
- concurrencia
- distribución del software en la plataforma

Gracias a la arquitectura logramos la *integridad conceptual* necesaria para dirigir nuestro proceso. La importancia de esta abstracción hace necesaria la definición de un rol que participe activamente en su creación y que ya ha sido detallado con anterioridad, el Arquitecto.

Arquitectura Ágil

A continuación detallaremos cuál es el propósito de la arquitectura en el marco del proceso ágil propuesto, cómo se construye y cómo se mantiene.

En este punto, y a diferencia de la mayor parte de las metodologías ágiles, AgEnD propone definir una arquitectura candidata en forma temprana. La idea subyacente de este patrón deriva de analizar la curva del costo de cambio propuesta por Barry Boehm originalmente. En la misma, Boehm indicaba que el costo de reparar un defecto descubierto durante la fase de requerimientos resultaba 200 veces más barato que reparar el mismo defecto en la fase de mantenimiento. Esto está graficado en la Figura 021, que muestra cómo se obtienen ahorros en una relación de hasta 200:1 al encontrar errores en fases tempranas versus a encontrarlos en fases más tardías. Este análisis llevo a Boehm a proponer el Modelo en Espiral, que mitigaba en forma temprana los riesgos de introducir defectos en la fase de requerimientos mediante la realización de sucesivas iteraciones de validación con el cliente.

Sin embargo, detrás de esta premisa estaba la idea de tener un entendimiento global de todos los aspectos funcionales de la aplicación, creando documentos de especificación completos, detallados, y con todas las normas de calidad requeridas. Esto terminaba degenerando el concepto formulado por el modelo en cascada que una vez que los requerimientos estaban bien definidos la construcción proseguía como una secuencia de fases. Ya han sido descritas todas las falencias de este modelo, por lo cual ahora se analizará como sería la curva del cambio en el universo de las metodologías ágiles.

Esta misma noción aplicaba a la idea de realizar cambios a los requerimientos. Dadas las características humanas que hacen muy difícil las actividades de relevamiento debido a que el cliente va cambiando aprendiendo lo que desea con el tiempo y derivando de la Figura 021 ya mencionada, se tomaba como referencia en las metodologías tradicionales la Figura 022 que mostraba el impacto del cambio en cada fase. De hecho era el mismo modelo en cascada o secuencial el que contribuía a que todos los cambios siguieran esa curva dado su escasa tolerancia al cambio. Esto fomentaba nuevamente la idea de que realizando una exhaustiva fase inicial de requerimientos el sistema tendría un mínimo nivel de cambios. Este supuesto era una falacia como muestra la evidencia propuesta por Capers Jones presentada en la Figura 023. Según este análisis el cambio a los requerimientos en proyectos de software va desde un 10% sobre el alcance total en los proyectos más chicos, hasta un 35% o más en los proyectos más grandes. Queda demostrado que la construcción de software se halla en un dominio de alto cambio y el proceso en cuestión debe tomar esto en cuenta.

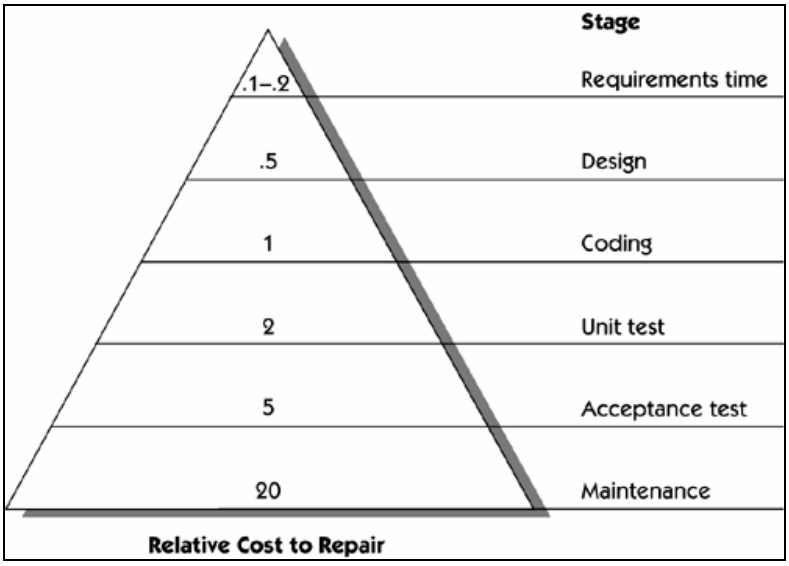

Figura 021. El costo relativo de reparar un defecto en diferentes fases del ciclo de vida. Tomada de [Leffingwell, 2001].

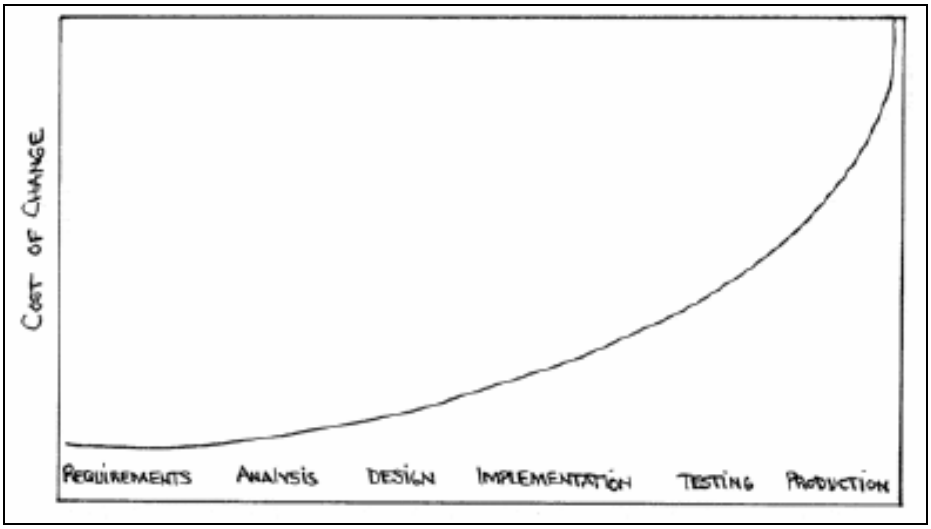

Figura 022. Curva del costo de cambio a los requerimientos en diferentes fases del ciclo de vida. Tomada de [Beck, 2000].

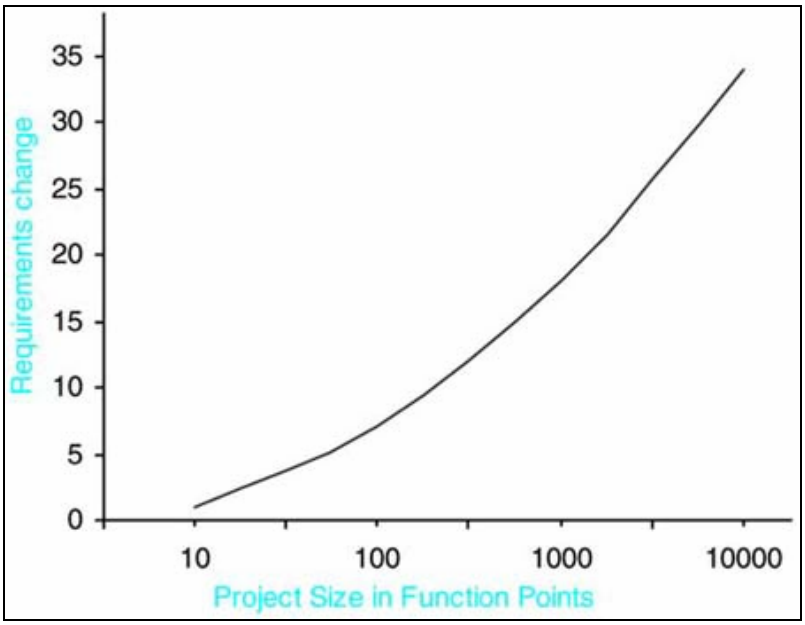

Figura 023. Tasas de cambio a los requerimientos en proyectos de software respecto al tamaño funcional de los mismos. Tomada de [Larman, 2003].

La premisa propuesta por AgEnD consiste en tener un proceso ágil que tenga alta tolerancia al cambio y que permita que la curva del costo de cambio propuesta por Boehm no escale exponencialmente en la mayoría de los casos. AgEnD sugiere que en los proyectos se tenga una gráfica como muestra la Figura 024. En la figura hay dos curvas: la número 1 que crece exponencialmente al igual que la original de Boehm y en la que se manifiestan los cambios relacionados con aspectos que involucran altos riesgos como la arquitectura candidata del sistema, mientras que en la curva número 2 que es bastante achatada pondríamos todos los cambios relacionados a la funcionalidad del sistema los cuales pueden ser absorbidos en las distintas iteraciones mediante el feedback continuo con el cliente.

La idea de AgEnD es tratar de mover todos los cambios a la curva número 2 (idealmente ese el postulado de XP, que plantea que la curva del costo de cualquier cambio es la curva número 2), pero reconoce que existen distintas aspectos que implican un alto costo al cambio una vez que se avanza en el desarrollo. Para aquellos factores que revisten las características de la curva número 1 AgEnD sugiere una definición previa al inicio del desarrollo en gran escala y establece al Documento de Arquitectura como el repositorio de estas cuestiones.

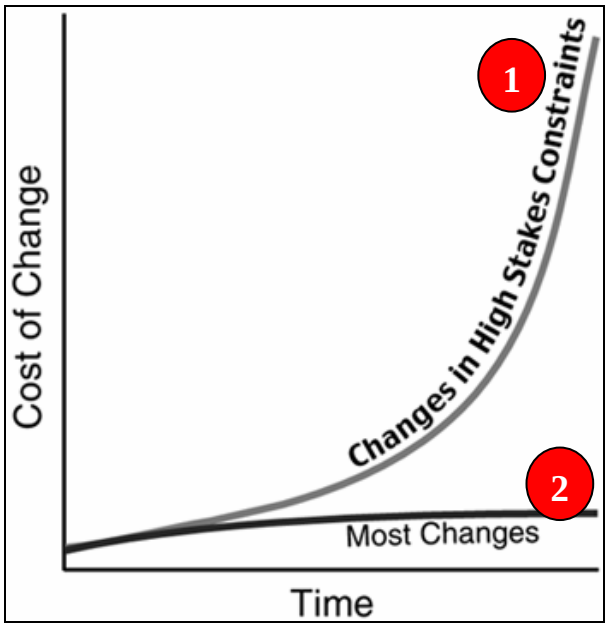

Figura 024. El costo del cambio actualizado para una metodología ágil. Tomada de [Poppendieck, 2003].

Sin embargo, dadas las características del proceso resulta burocrático desarrollar un documento de grandes proporciones que guíe el proceso con las decisiones más importantes del diseño, que incorpore extractos de los diversos aspectos que son reconocidos como esenciales para el desarrollo y que sea actualizado en cada iteración para servir a su propósito.

Pero antes de esto deberíamos analizar, siendo que el propósito de la arquitectura es la comunicación de aspectos esenciales del software entre las partes involucradas o stakeholders, ¿no se ve reducido drásticamente su alcance dada la estrecha comunicación propuesta por la metodología? En otras palabras, al combinar el principio de *Máxima Comunicación* con el de *Arquitectura Ágil*, debemos cambiar la definición de arquitectura tradicional ya que la misma no coincide con las prácticas propuestas.

El principio de *Arquitectura Ágil* en AgEnD consiste en mantener una visión única, formulada por el equipo de proyecto de las decisiones consideradas críticas para el desarrollo. Será el mismo equipo el encargado de construir, mantener y obtener los beneficios de la misma. En consecuencia, la arquitectura no va a ser un documento estático con una estructura determinada a la cual el equipo se deba ajustar aunque no contribuya al desarrollo. Irá evolucionando con el desarrollo de los componentes y será mantenida hasta el punto que contribuya a la comunicación de los aspectos tecnológicos dentro del grupo. Una vez que el conocimiento esté en las mentes de todos, la misma habrá cumplido su función y el documento podrá ser dejado a un lado. Esto es así

debido al principio de *Proceso Orientado a las Personas*, el cual describe la necesidad de construir aquellos artefactos indispensables para el proyecto de software, aquellos que sirvan algún propósito claro.

Al momento de definir la arquitectura, AgEnD propone una serie de modelos extraídos y extendidos a través de estereotipos del Unified Modeling Language (UML). Los mismos sirven como guía según el tipo de aplicación que se este construyendo. En general, nos referiremos a aplicaciones que se encuentren dentro del espectro de aplicabilidad de AgEnD; es decir, aplicaciones del sector IS/IT, de una escala media o pequeña, con alto grado de involucramiento de los usuarios.

Estos modelos están especificados UML y sirven para describir distintas vistas de la arquitectura necesarias para la comunicación dentro del equipo. Entre los modelos que el arquitecto debería manejar al momento de definir la arquitectura encontramos:

- Diagrama de Despliegue (Figura 025 y 026):

  Definición: el diagrama de despliegue muestra como y donde el sistema será desplegado. En el mismo se muestran nodos físicos, procesadores, y los componentes que corren en cada uno. Como se observa en las figuras este diagrama puede tener distintas implementaciones de acuerdo al grado de detalle que se quiera llegar. Sin embargo, es muy recomendable ya que muestra como la aplicación será llevada al ambiente de producción mediante la separación de los componentes en nodos.

- Diagrama de Componentes (Figura 027):

  Definición: el diagrama de componentes muestra las piezas de software o entidades que conforman al sistema; en general, un componentes es de más alto nivel que una clase y suele ser implementado en tiempo de ejecución por un número de clases.

- Diagrama de Secuencia (Figura 028):

  Definición: el diagrama de secuencia es una representación de las interacciones entre clases y objetos para llevar a cabo una operatoria a lo largo del tiempo. Se utiliza para mostrar flujos de trabajo, pasaje de mensajes y la cooperación necesaria para brindar un determinado resultado.

- Diagrama de Estado (Figura 029):

  Definición: el diagrama de estado sirve para mostrar como un elemento (clase) cambia de estado a lo largo del tiempo y las transiciones que le son permitidas en cada caso con las correspondientes condiciones.

- Diagrama de Clases (Figura 030):

  Definición: el diagrama de clases captura la estructura lógica del sistema – las clases y/o entidades que componen al modelo. Es considerado un modelo estático ya que muestra las clases existentes y los atributos, métodos de cada una.

- Diagrama de Datos (Figura 031):

  Definición: el diagrama de datos sirve para mostrar como el modelo será persistido en un RDBMS. El mismo está dentro del perfil de UML para modelado de datos y maneja estereotipos como Tabla, Primary Key, Foreign Key, etc.

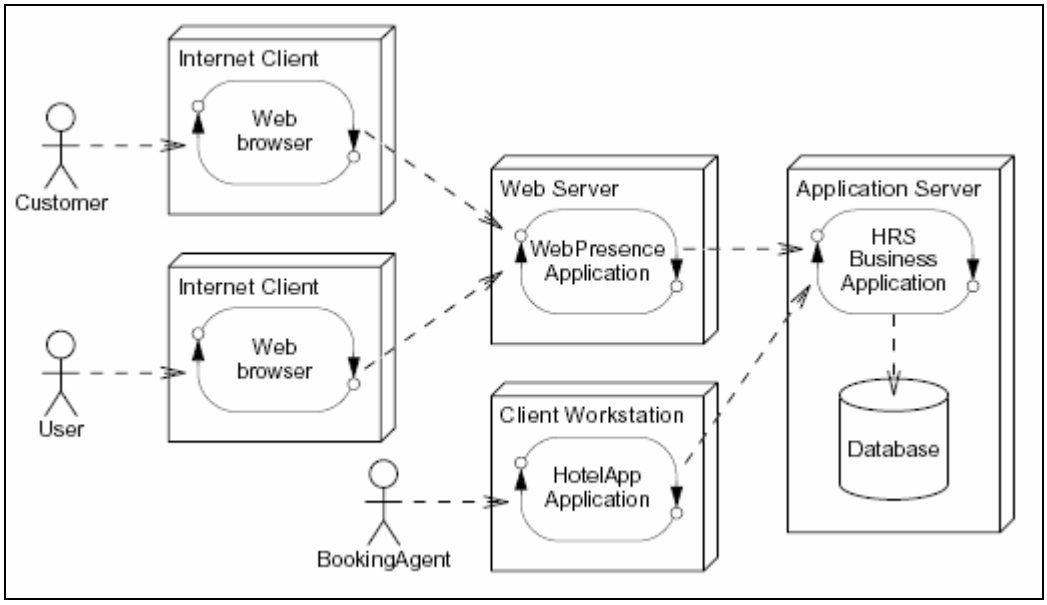

Figura 025. Diagrama de Despliegue de alto nivel en UML. Tomada de [Sun, 2003].

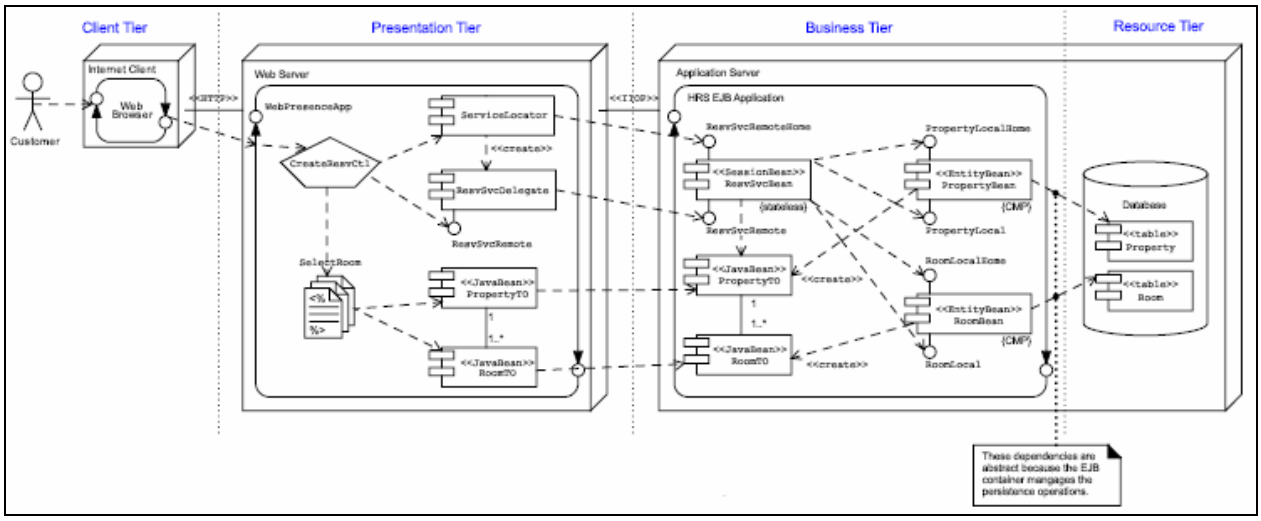

Figura 026. Diagrama de Despliegue detallado en UML. Tomada de [Sun, 2003].

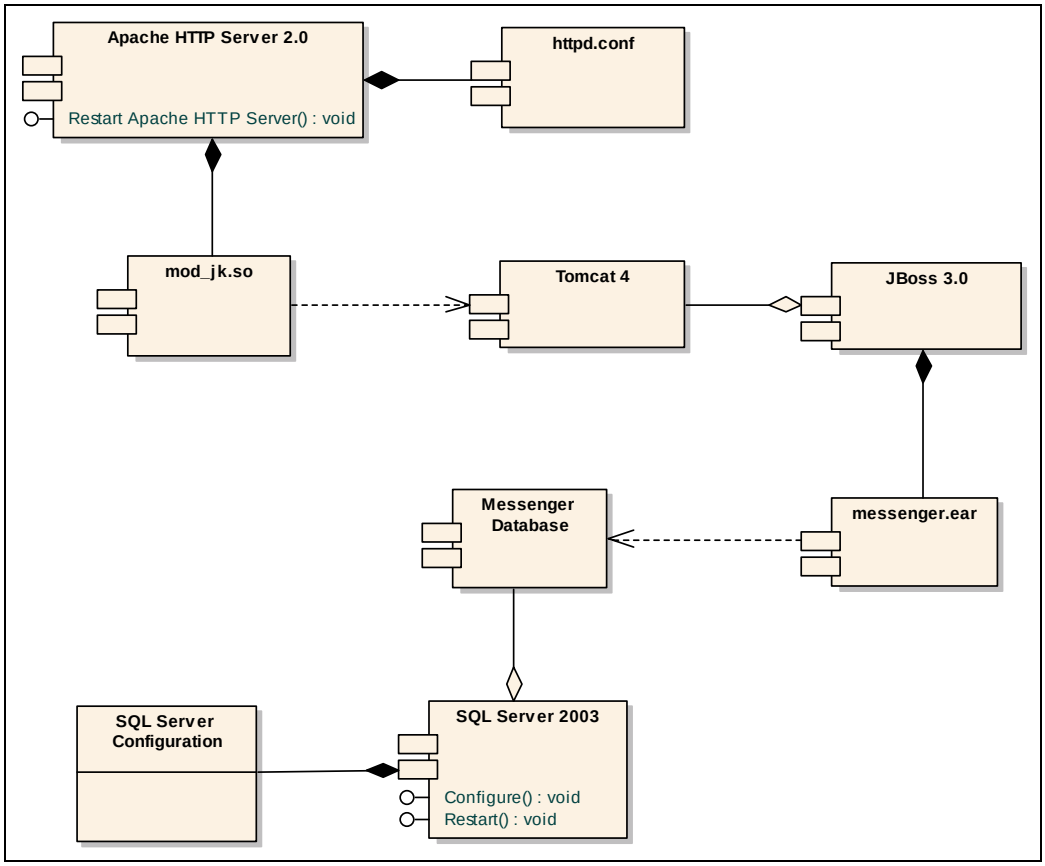

Figura 027. Diagrama de Componentes en UML. Tomada de [EA, 2003].

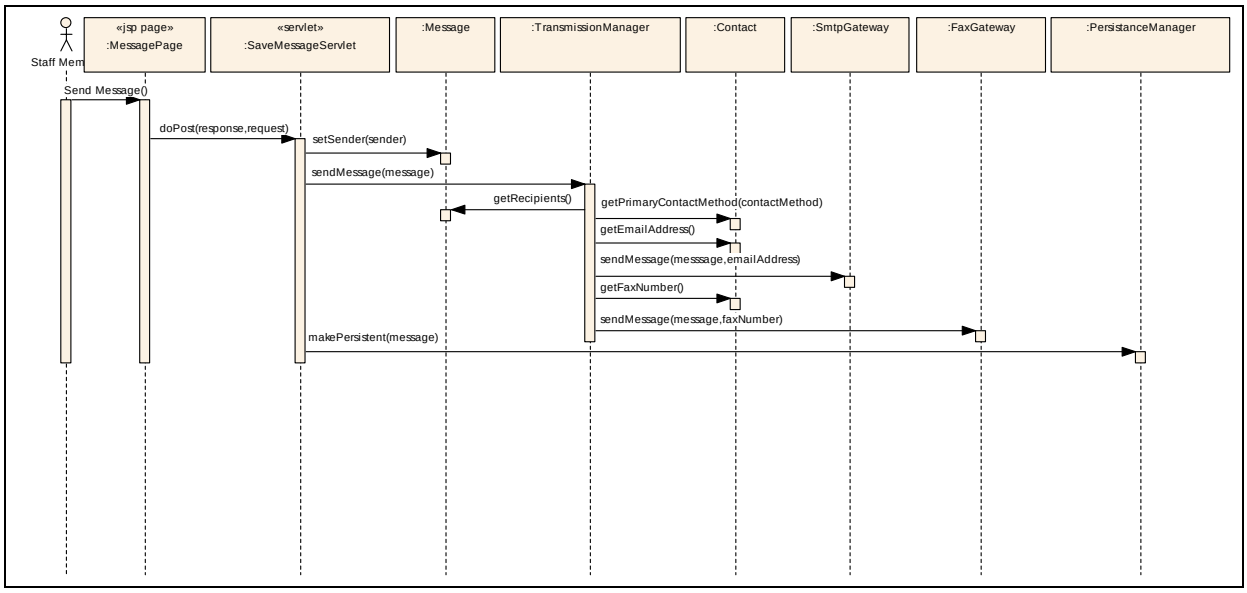

Figura 028. Diagrama de Secuencia en UML. Tomada de [EA, 2003].

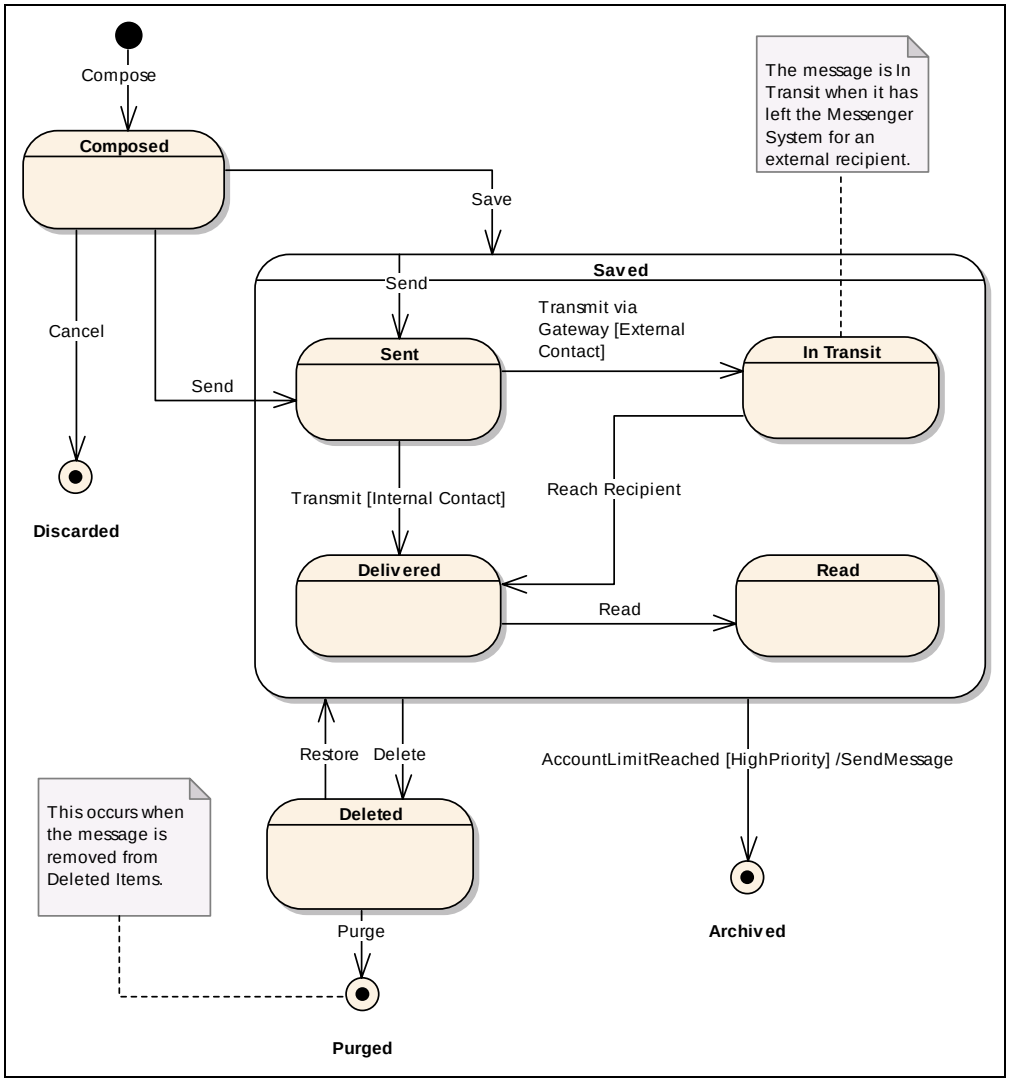

Figura 029. Diagrama de Estado en UML. Tomada de [EA, 2003].

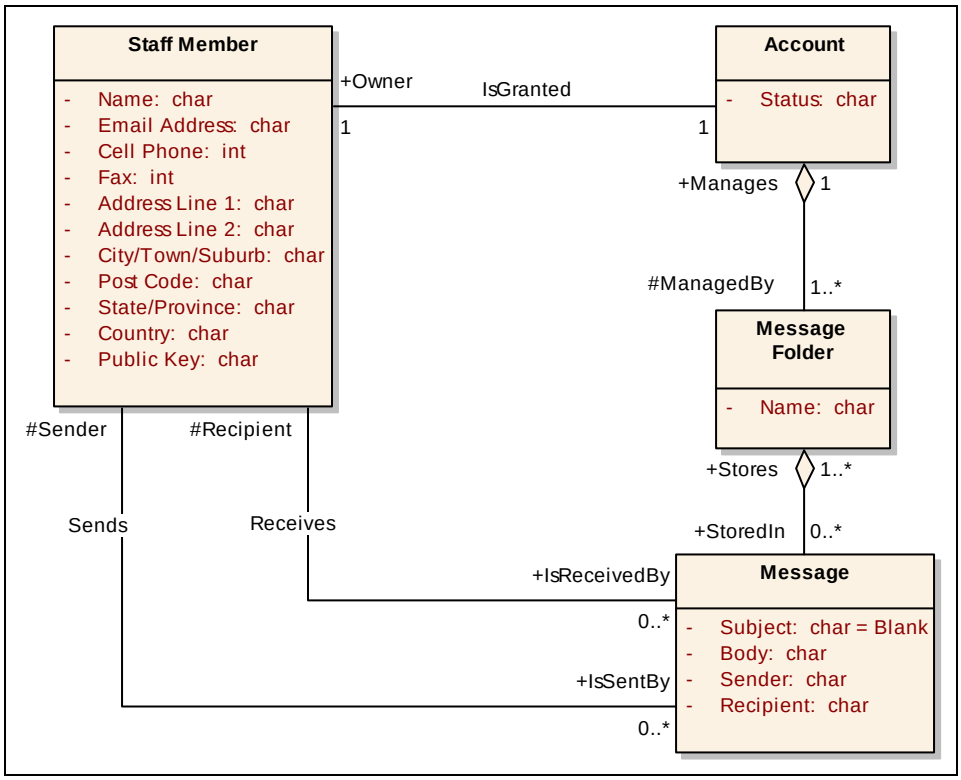

Figura 030. Diagrama de Clases en UML. Tomada de [EA, 2003].

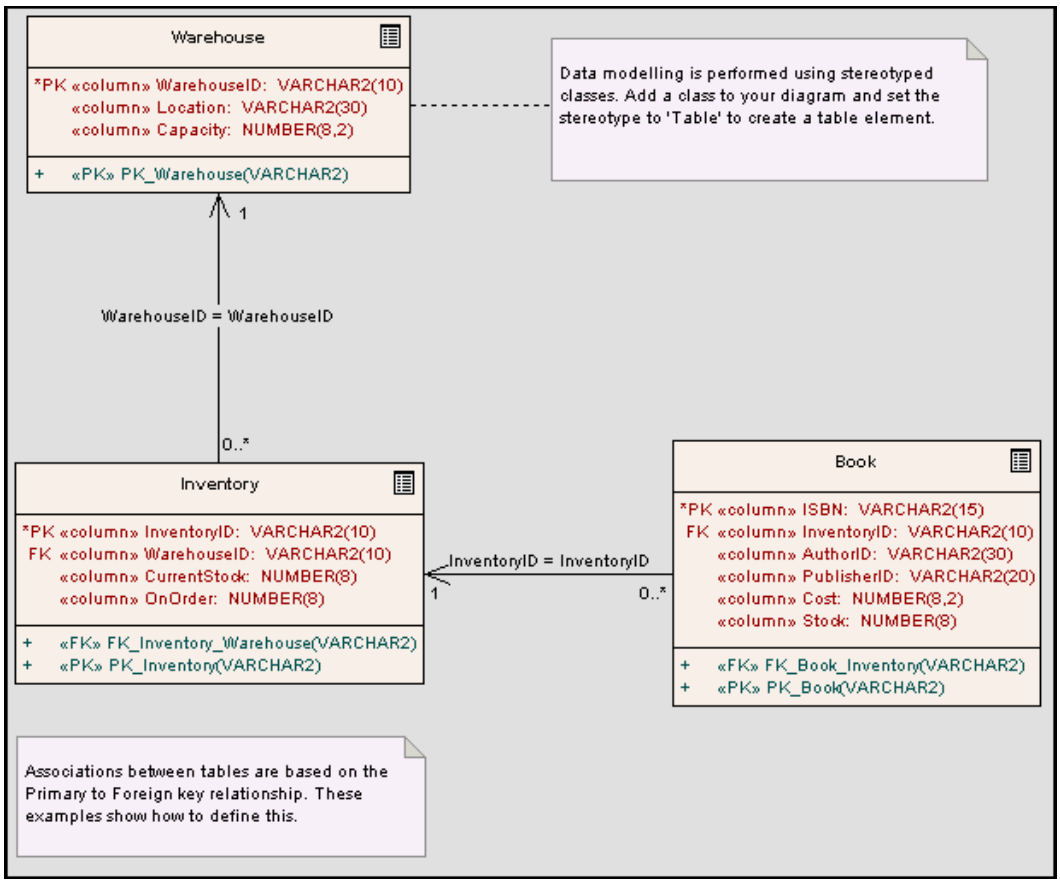

Figura 030. Diagrama de Datos en UML. Tomada de [EA, 2003].

Se debe tener en cuenta que debe existir un balance entre el esfuerzo en realizar estos diagramas y la posibilidad de comunicar dichas ideas en forma directa. De acuerdo al patrón de *Máxima Comunicación* conviene que estos diagramas estén en algún radiador de información ya sea mediante impresiones en hojas grandes colgadas, en una intranet de administración de conocimiento, o simplemente dibujados en pizarrones en los casos que se desee más informalidad. Una vez que el propósito comunicacional es servido no tiene sentido seguir detallando y/o manteniendo estos diagramas.

Integración Continua

Indudablemente una de las prácticas que hoy se encuentra recomendada por la mayor parte del espectro de metodologías modernas (no solo ágiles) es el de *Integración Continua*.

Esta práctica nace a partir de los fracasos acarreados por las fases de integración de los modelos tradicionales en que una vez que se tenían todos los módulos construidos se iniciaba una intensa y compleja fase en que los mismos se integraban unos con otros hasta lograr el producto a ser entregado. Era en dicho lapso en que ocurrían los mayores descalabros en materia planificación, excediéndose de manera desmedida el tiempo, haciendo que fracasen proyectos que habían concluido con éxito sus fases anteriores. Una de las causas principales consistía en las complejas interacciones existentes entre los distintos módulos, construidos por distintos equipos, los cuales al ser integrados por vez primera presentaban bugs que eran muy difíciles de encontrar por estar generados por distintas personas.

La industria del software aprendió su lección, y ya desde hace algún tiempo, la *Integración Continua* es considerada esencial en cualquier proceso moderno de desarrollo por más burocrático o minimalista que resulte. Por dar un ejemplo, la metodología eXtreme Programming contiene la *Integración Continua* entre sus 12 prácticas claves. Más aún es de público conocimiento que desde hace muchos años Microsoft viene utilizando esta práctica en su proceso de desarrollo de donde surgió la noción del “daily build”, en que todas las noches se integraba, compilaba y linkeaba el código generado para ser luego probado por los casos de prueba automatizados. Steve

McConnell [McConnell, 1996] fue uno de los que más analizó esta técnica recomendándola entre sus conocidas best practices. Martin Fowler [Fowler, 2002] escribió un artículo bastante descriptivo de los beneficios de esta práctica.

La *Integración Continua* en AgEnD consiste en poseer un ambiente automatizado mediante el cual se pueda realizar el check out de la herramienta de Administración de la Configuración que se este utilizando y posteriormente tener un entorno que tome el código fuente, genere los builds con todos los pasos intermedios que esto involucre, corra los suites de tests automatizados, y emita un reporte con el resultado del proceso. El énfasis está puesto en la automatización del proceso de manera que resulte algo lo más transparente posible para el equipo de desarrollo. Existen en la actualidad herramientas muy potentes – en algunos casos, gratuitas por ser de código abierto – las cuales permiten que esta práctica pueda ser llevada a la realidad con un mínimo esfuerzo. Un ejemplo de herramienta Open Source de estas características es el **Cruise Control** que funciona en conjunto con **Ant**, **CVS**, y **JUnit** para llevar a cabo los builds continuos en una máquina.

Para tener un build exitoso es recomendable que el mismo tenga ciertas características que nos den seguridad respecto a la consistencia e integridad del mismo. En un entorno ideal un build solo podrá ser llamado exitoso si se dan todos los siguientes pasos correctamente sin intervención humana:

- los últimos fuentes han sido bajados del repositorio de versionado
- cada archivo fue compilado de cero
- los ejecutables han sido linkeados y empaquetados para su despliegue
- el sistema es ejecutado y las pruebas automatizadas son disparadas

Para entender la práctica en cuestión se dará un ejemplo concreto de su aplicación en un proyecto de software usando una metodología como AgEnD. Se toma el supuesto de que se tiene un equipo de desarrollo en que cada programador se encuentra programando un componente en forma paralela a los demás. A intervalos regulares cada programador terminará la construcción de su componente, correrá las

pruebas unitarias sobre el mismo[7] y una vez que pase exitosamente el 100% de las mismas procederá a integrarlo. Cuando el programador se disponga a integrar lo que ha construido realiza previamente una actualización de la copia local de su repositorio para chequear que esté usando la última versión de todos los ítems de configuración y luego subirá sus cambios mediante un check-in al repositorio.

En casos que no se cuente con una herramienta de builds automatizada, el programador hará un check out del módulo correspondiente en la máquina que ha sido designada como repositorio de los builds maestros. En la misma ejecutará el script de deployment[8] que se encargará de compilar y empaquetar las clases para su despliegue. Suponiendo que el resultado del script es exitoso se podrá efectuar el test funcional en caso que se desee liberar la versión. Una vez que el programador deja la máquina de integración tendrá una sensación de logro y una inmediata recompensa de un trabajo bien realizado.

Sin embargo, deberemos recordar que hasta este instante estamos verificando la sintaxis y semántica del código, pero no sabemos nada respecto a si cumple las necesidades del usuario. Para esto se tienen las pruebas funcionales automatizadas o manuales que deberá crear o especificar el tester junto con el cliente.

Peopleware

Una de las cuestiones en que se enfoca AgEnD es Peopleware - esto es consistente con la idea de estar alineada bajo el *Agile Manifesto* ya mencionado. Este término refiere a las personas involucradas en el desarrollo de software y a todos los aspectos relacionados con las mismas.

Las primeras nociones de Peopleware surgieron alrededor de 1969, cuando la industria del software apenas comenzaba a madurar, de manos de Gerald Weinberg en su libro *The Psychology of Computer Programming*, posteriormente reeditado en 1998 [Weinberg, 1998]. En el mismo, Weinberg relativizaba los aspectos relacionados con el proceso y las técnicas utilizadas para construir software abocando la importancia que

[7] Esta idea surge del framework de testing creado por Kent Beck para Smalltalk, posteriormente portado y popularizado por la versión en Java, denominada JUnit, desarrollada por Beck y Erich Gamma. Más información en http://www.junit.org

tenía el factor humano en el mismo. La comunidad informática no incorporó las consideraciones de Weinberg, quien siguió escribiendo acerca de estos temas durante el tiempo subsiguiente. Pasaron algo más de 15 años hasta que Tom DeMarco tomara nuevamente estas ideas para su libro *Peopleware: Productive Projects and Teams* [DeMarco, 1987] también reeditado en 1999. El libro de DeMarco tuvo mayores repercusiones en la comunidad que las que había tenido su predecesor, aunque no logró imponer el cambio necesario para considerar las cuestiones humanas como dominantes en el desarrollo del software. Cabe destacar que algunas empresas sí tomaron las prácticas de DeMarco con algunos resultados bastante exitosos – el ejemplo más destacable es Microsoft.

Llevando las ideas de todas estas fuentes al contexto de esta tesis, AgEnD toma a las personas como partes esenciales del desarrollo y las encuadra en un marco humano necesario para el entendimiento del impacto que las mismas tienen en la construcción de los diversos artefactos. Dicho marco humano se centra en tres áreas de estudio: organización, área de desarrollo e individuo.

Dentro de la organización agrupamos aquellas cuestiones externas al área de desarrollo que no son controladas por el mismo.

- El lugar de trabajo.
- Recursos de tecnología disponibles.
- La cultura de la empresa.

Por área de desarrollo entendemos la forma en que se relacionan y organizan los individuos para realizar el desarrollo.

Dentro del análisis del individuo se encuentran los factores que afectan a un individuo al momento de desarrollar software. Las personas necesitan tener un propósito para lo que hacen y esto toma mayor importancia en relación a su trabajo ya que éste ocupa en promedio el 35% del tiempo durante el que realizan actividades. Consecuentemente, se debe fomentar que el individuo esté cómodo en su trabajo y que exista una buena relación con sus pares. Para esto es de suma importancia que el Líder de Proyecto monitoree las dinámicas que entran en juego en el grupo así como el comportamiento de los individuos.

[8] Una de las herramientas más populares para despliegue en la plataforma Java es el ANT. Este es un subproyecto dentro de la comunidad de Jakarta que brinda una herramienta indispensable para despliegue automatizado de aplicaciones complejas. Más

La cultura de una empresa se define como el conjunto de ideologías, valores, metas, que perciben los individuos que conforman la misma. Representa el folclore de las personas que interactúan bajo una misma organización. Una de las primeras cuestiones con las que debe inmiscuirse rápidamente un individuo al comenzar a formar parte de una empresa es su cultura. Gracias a ésta el empleado se irá adaptando, cambiando sus valores, formulando nuevos juicios y terminará poniéndose las mismas metas que el resto de las personas o abandonará el intento por no sentirse cómodo con la misma. La cultura de la empresa tiene tanta relevancia como el trabajo en sí para el que fue contratado el empleado.

Para analizar cómo impacta este factor en el desarrollo de software, utilizando AgEnD como nuestra metodología, deberemos comenzar por analizar la cultura de la empresa en que vamos a implementar el proceso. La empresa ideal para el éxito de un proceso ágil es aquella que comparte un principio de *information sharing* versus *information keeping*. Entendemos por *information sharing* la aptitud de las personas unidas bajo un mismo objetivo de colaborar para que el conocimiento sea transferido de donde está contenido a donde es requerido. Esta noción se contrasta con la *information keeping* en la que un grupo de personas compite por acumular conocimiento y se esfuerza en no comunicarlo a quienes lo necesiten.

Estos extremos de una recta representan el grado de madurez de la disciplina de Administración de Conocimiento (KM) dentro de la organización. Entendemos por KM al proceso organizacional para retener, organizar, compartir, y actualizar conocimiento crítico para la performance individual y la competitividad organizacional [Davenport, 1998]. A continuación expondremos esto en más detalle.

Calidad en el Proceso

El referirnos a AgEnD como una metodología ágil no quiere decir que el énfasis puesto en los entregables y la rapidez del desarrollo no tengan en cuenta el problema de la calidad.

Tomando como marco de referencia al CMM es sabido que la mayoría de las empresas del país se encuentran en el nivel 1, denominado *Inicial*. Una de las razones

información en http://www.ant.org

por las cuales esto ocurre radica en la falta de un proceso de desarrollo maduro y en la tendencia inherente en la gente de sistemas de seguir el modelo *Codificar y Probar*. Esta tendencia se debería modificar de forma de seguir un proceso relativamente disciplinado, generando documentación de análisis/diseño previamente, haciendo el testing luego de la codificación, siguiendo entregas definidas. Sin embargo, al momento en que el tiempo apremia se dejan de lado todas estas prácticas, desapareciendo el proceso al punto de volver al caos del que se deseaba escapar.

Una de las razones de este fracaso consiste en la falta de convencimiento pleno por parte del equipo de desarrollo en el proceso que utilizan diariamente. Mientras los hitos se vayan cumpliendo siendo las entregas aceptadas por los clientes el equipo se siente confidente de que su proceso los llevará al éxito y que además lo están siguiendo correctamente. Todos comentarán la necesidad de mantener un proceso disciplinado mostrando las experiencias pasadas como demostración de los resultados y de los productos entregados.

Podrá llegar el momento en que el proyecto comience a retrasarse en las entregas, a disminuir la calidad o a remover funcionalidad de las mismas. Será en estos momentos en que nacerá un sentimiento de desesperación que comenzará en el management descendiendo en la jerarquía hasta el equipo de desarrollo. Se pedirán entregas imposibles, las cuales llevarán a los programadores a codificar en forma masiva, dejando de lado completamente cualquier viso de proceso, erradicando completamente la calidad del software construido.

Por estas cuestiones es importante que el proceso se tenga en alta estima y que la gente esté convencida de su utilización. En caso contrario, el proceso no cumple su función. Para mantener la salubridad del mismo incluimos en la próxima sección una serie de disciplinas que permiten tener siempre un proceso de calidad que se adecúe a las necesidades de la gente que lo utiliza.

Enfoque Sistémico

Uno de los motivos principales de desarrollo de esta tesis consiste en una investigación extensa en los campos de la psicología sistémica, la administración de empresas y las relaciones laborales; para incorporar el conocimiento de estas áreas en el desarrollo de software.

Ambiente de Desarrollo de Software

Analizaremos el Desarrollo de Software enfocándonos en tres cuestiones que hacen al ambiente en el que se lleva a cabo un proyecto. Las tres perspectivas en cuestión refieren a las diversas áreas de interacción entre las personas que estén afectadas en cualquier grado al desarrollo.

La primera perspectiva refiere a la Cultura Organizacional (CO), en la que analizaremos como los individuos realizan su trabajo dadas las condiciones externas al equipo del desarrollo: el resto de la Organización, el Cliente con que se trabaja, los stakeholders del proyecto y todas aquellas interacciones que salgan fuera del alcance de los desarrolladores.

La segunda perspectiva refiere a la Equipo de Desarrollo (ED), en la cual analizaremos en detalle cómo los distintos roles establecidos por la posición de cada individuo en la empresa influyen en el resultado y la performance de todo el equipo. Describiremos los tipos de equipos que se pueden plantear, los problemas sistémicos que pueden acaecer en los mismos, y cómo la situación particular del desarrollo de software nos obliga a centrar nuestra atención en muchos aspectos de las relaciones entre los participantes.

La tercera y última perspectiva planteada es la de la Psicología de Individuo (PI), en la cual estudiaremos cómo el individuo internaliza los procesos que entran en juego al realizar las distintas actividades que abarca el desarrollo. Desde el Modelado del Negocio hasta la Administración de la Configuración cada disciplina requiere características propias del individuo así como existentes características que trascienden las tareas y que son esenciales para que una persona se pueda embeber en el dinámico mundo de los proyectos de desarrollo de IS/IT. La intención es mostrar las aptitudes

necesarias para cada rol, y las distintas formas en que estas contribuyen a que la persona este satisfecha de su trabajo.

Para encarar este estudio haremos una breve introducción al cuerpo de conocimiento que nos provee la Psicología Sistémica. Esta rama de la Psicología, surgida a partir de cuatro pilares surgidos el siglo pasado que dieron el comienzo de la Cibernética y la Informática. Las ideas de Wiener en relación al feedback, y el pensamiento circular surgidas a principios del siglo pasado fueron uno de las primeras influencias de la sistémica. Posteriormente, Turing trajo consigo el trabajo necesario para atacar la complejidad inherente a cualquier sistema mediante modelos que servían para reducir dicha complejidad y hacerla comprensible para la mente humana. A partir de esta teoría se comenzaron a atacar los problemas mediante el desglose sistémico en distintos modelos que reflejaran las partes que se iban a estudiar. Fue Shannon el encargado de construir la teoría de la información a partir de la cual se vio la posibilidad de transmitir cualquier tipo de información a través de un canal de datos. Ese sería el inicio de lo que actualmente denominamos la globalización en materia de comunicaciones. Por último, Von Neumann trabajó en formas de convertir la información en señales eléctricas de manera que las mismas pudieran ser procesadas en forma automática. De esta forma, se sentaron las bases para la creación de las computadoras que procesarían la información que se les cargaba sin requerir de operatoria manual.

Gracias a estos cuatro marcos teóricos se configura la Teoría General de Sistemas, la cual ha cambiado la forma de estudio de todas las ciencias desde las humanísticas hasta las científicas. En particular, surge una nueva disciplina que es la Informática que está relacionada con las actividades que conciernen a los sistemas de información. Es decir, es la consecución natural de la Teoría General de Sistemas y su aplicación a todas las actividades humanas en las que existe algún manejo de información.

Es notable el aporte de Roman Jakobson, uno de los creadores de la lingüística moderna que traslada los postulados de la teoría de la información de los sistemas a lo que se denomina la teoría de la comunicación. El modelo planteado por Jakobson, en que en toda comunicación existe un Emisor, un Receptor y un mensaje transmitido sienta las bases para la especificación de las interacciones entre las personas como un fenómeno sistémico susceptible de ser analizado. Retornando a la Psicología, estas

nociones son utilizadas por diversos psicólogos que estudian la teoría de la comunicación y como ésta influye en patologías y en la forma en que los seres humanos perciben la realidad. Dentro de este contexto surge la denominada ciencia cognitiva que intenta describir cómo se construye el conocimiento en una organización.

En particular para esta tesis, nos interesa el análisis de lo que la psicología sistémica denomina el "efecto paradigma". El concepto toma sus ideas de los textos de Kuhn en relación a los movimientos de la ciencia, en los que en un momento dado se reconstruye todo el conocimiento a partir de un nuevo paradigma. Ejemplo de esto lo tenemos a fines del siglo XIX cuando se realiza el experimento conocido con el nombre de la catástrofe del ultravioleta que muestra como el paradigma propuesto por la mecánica clásica no alcanza a explicar las interacciones a niveles subatómicos. Esto deriva en una revolución que termina de ser afirmada con la teoría cuántica descripta por Planck y Heisenberg, entre otros, y la relatividad propuesta por Einstein. Sin embargo, la transición no fue gradual y requirió de varias décadas hasta que el nuevo paradigma tuviera status en la comunidad científica.

Existe una analogía de este efecto en relación a las personas que componen una organización y tiene que ver con la resistencia al cambio. Al momento de implementar una metodología la gente posee un cierto conocimiento respecto a sus actividades y la forma en que se realizan. La idea es que de alguna forma la implementación debe cambiar el modo de pensar de las personas modificando el paradigma metodológico existente en la organización. Diversos estudios muestran que cuando un paradigma cambia, todo el mundo vuelve a cero; es decir, el conocimiento es reestructurado. Por lo tanto se debe tener especial cuidado al momento de llevar a cabo este cambio analizando la posición de cada individuo y la tolerancia al cambio de la organización.

Tom DeMarco nos presenta un continuo que coloca a las personas en una escala que va de los "Lealmente Ciegos" hasta los "Opuestos Militantemente", pasando en el medio por los "Creyentes pero Escépticos" – ver Figura 031. Es importante identificar como es la posición de los individuos en este espectro de manera de llevar a la organización en un camino de pequeños cambios hasta llegar al nuevo status quo deseado.

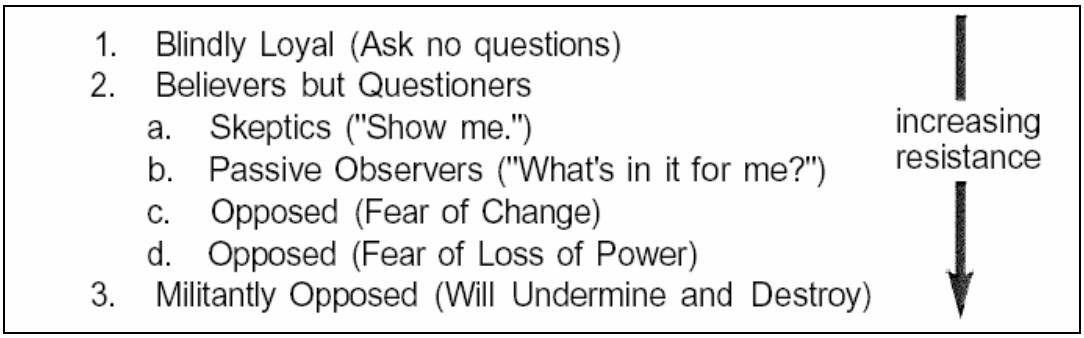

Figure 031. El Continuo de la Resistencia al Cambio. Tomado de [DeMarco, 1999].

Martín Wainstein, titular de cátedra de *Teoría y Técnica de la Clínica Sistémica* de la Facultad de Psicología de la Universidad de Buenos Aires, expone un mecanismo utilizado para llevar a cabo intervenciones sistémicas dentro de las organizaciones extensible a grupos de trabajo e individuos. El mismo tiene una sorprendente analogía al ciclo de mejora de proceso de calidad propuesto por Edward Deming a mediados del siglo pasado. Basado en el Bucle de Calais, la Figura 032 muestra los pasos existentes:

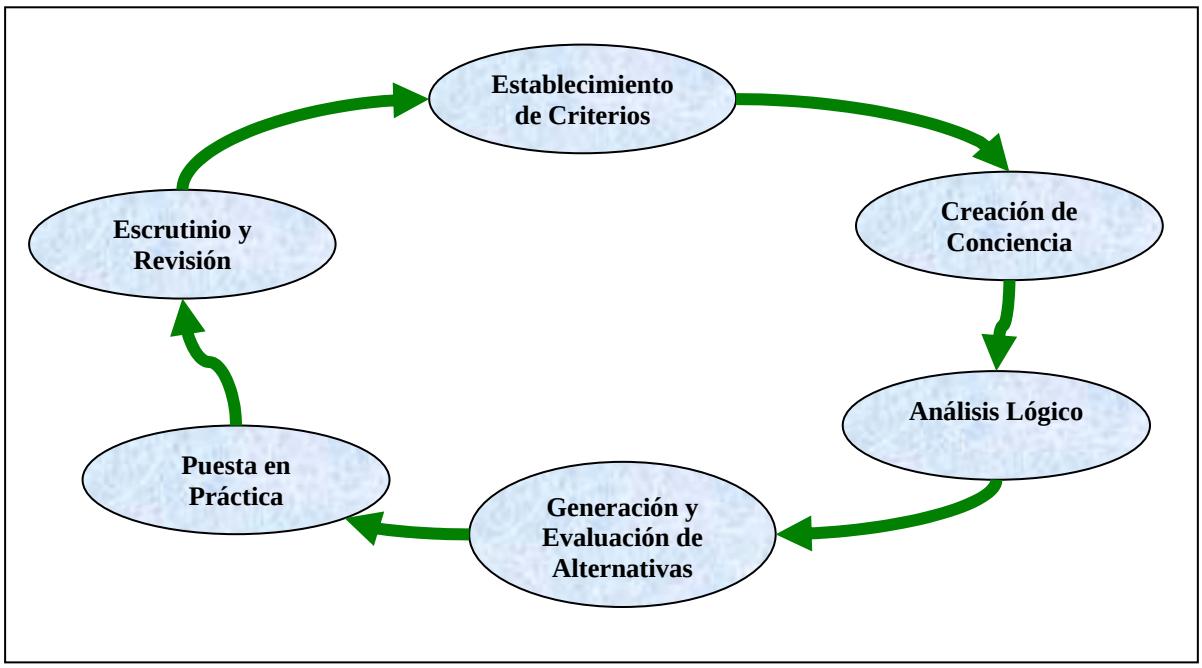

Figura 032. Metodología de intervención en organizaciones.

Esta figura identifica los pasos necesarios para un gradual ciclo de mejora de procesos. Se parte por el establecimiento de criterios en que las partes interesadas expondrán los problemas existentes que serán motivos de la intervención. Luego, se comenzará a crear una conciencia que facilite el cambio, para esto tomaremos el continuo mencionado anteriormente respecto a la posición de los individuos respecto al cambio. A continuación se efectuará un análisis de la situación, entendiendo las causas

detrás de los problemas para poder empezar a plantear soluciones. Se seguirá estableciendo alternativas que permitan dar soluciones a los problemas existentes. En forma conjunta se deberán formular criterios para evaluar si se ha llegado al éxito. El paso crítico consiste en poner en práctica las soluciones generadas. Finalmente, se evaluará el resultado de la puesta en práctica, haciendo una revisión que sirva para decidir sobre la continuación de iteraciones en pos de resolver el problema o si ya el status quo ha sido cambiado.

Tomaremos este procedimiento para establecer los pasos necesarios para implementar AgEnD en una organización.

Implementación del Proceso

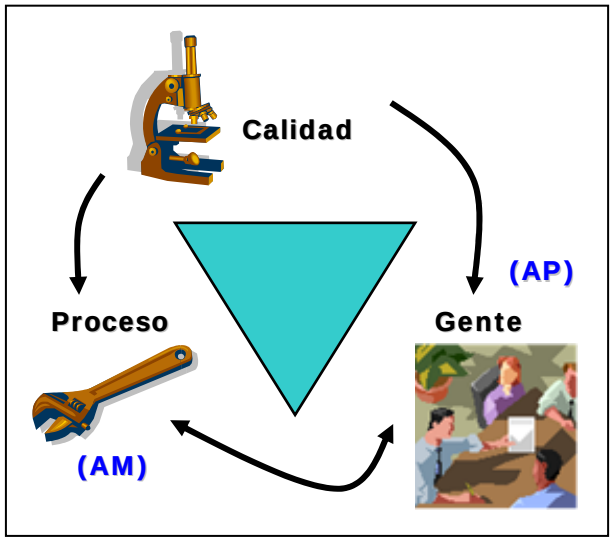

Figura 033. Las tres guías fundamentales de la Implementación de AgEnD.

Como se observa en la Figura 033, AgEnD agrega al Universo de Procesos Ágiles un framework para uno de los más grandes problemas que se presentan al elegir una nueva metodología: su implementación en la organización. Según la filosofía planteada en el *Manifiesto Ágil*, el proceso debe ser adaptado para las personas que serán usuarias del mismo. AgEnD sugiere dirigir el esfuerzo en tres actividades principales que son la **Adaptación Metodológica (AM)** que consiste en adaptar las diferentes partes del proceso a la realidad organizacional, la **Adaptación de las Personas (AP)** que consiste en institucionalizar el proceso en las personas que están involucradas en la utilización del mismo y, por último, la **Evaluación (EV)** que permite

ir midiendo el progreso de la implementación de forma de poder verificar el resultado. Para ello, se deberá tener especial cuidado de tomar métricas que permitan medir el avance en ambas actividades.

El Ciclo de Vida propuesto en la implementación de la metodología corresponde a un proceso iterativo en la que se van desarrollando las tres actividades antes mencionadas. En el ciclo se tienen iteraciones que desembocan en reuniones de revisión (que forman parte de las Actividades de la **Evaluación**) en las cuales se reporta el status de la implementación. En la Figura 034 se puede observar cómo a medida el proyecto avanza en iteraciones se inician las actividades de adaptación metodológica de AgEnD para que la misma sea acorde al contexto de desarrollo de la organización, del proyecto, y de las personas. Una vez que dicha metodología comienza a ser adaptada al entorno, se requiere una capacitación de las personas para que las mismas entiendan todas las actividades que deben realizar bajo AgEnD. En forma paralela se desarrollan las actividades de evaluación que otorgan el feedback necesario para el control y management de la implementación. El responsable primario de estas actividades es el Coordinador quien deberá supervisar y asesorar respecto a la implementación de AgEnD.

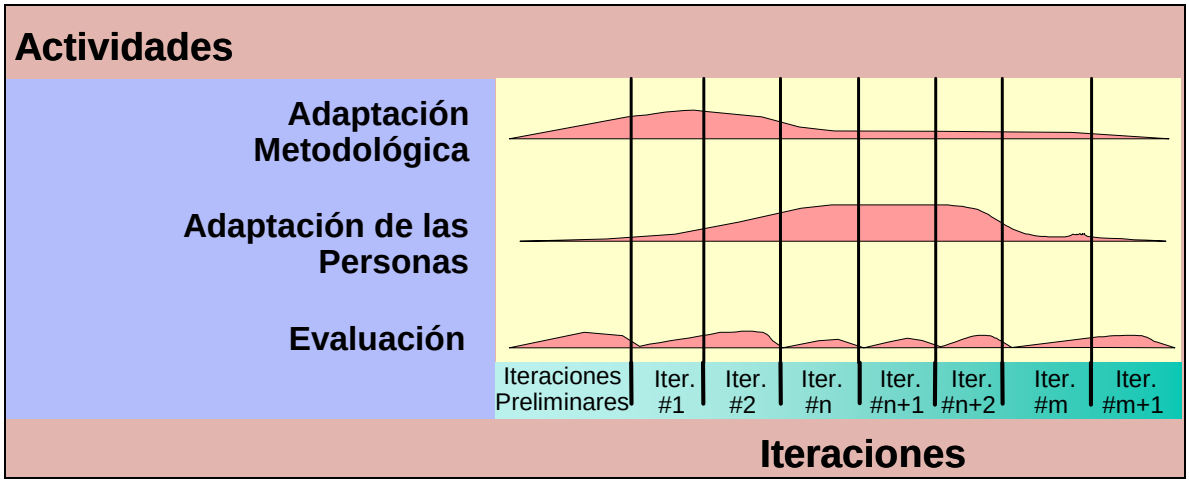

Figura 034. Actividades desarrolladas durante la implementación de AgEnD.

Adaptación Metodológica (AM)

La disciplina de Adaptación Metodológica consiste en adaptar el proceso tal cual ha sido expuesto en esta tesis a las necesidades de la empresa. Para esto se tomarán

las distintas partes de la metodología seleccionada evaluando cuáles se ajustan a la realidad organizacional y a su cultura.

Uno de los primeros pasos es relevar las características de los proyectos que se realizan en la empresa para determinar las necesidades metodológicas. En general se deberá tener en cuenta que los procesos ágiles aplican a realidades organizacionales particulares. AgEnD como fue mencionado se recomienda en aquellos casos en que se tienen recursos capacitados con buenos niveles de comunicación y con proyectos que pueden ser fragmentados en pequeños grupos de desarrollo (idealmente no más de 15 personas).

El siguiente paso consiste en capacitarse en la metodología ágil seleccionada; en el caso de AgEnD la información brindada por esta tesis y diversos papers presentados en distintas jornadas servirá de guía para el Coordinador. Una vez que se ha leído el material y que el Coordinador posee el know how necesario, el primer paso será el de adaptar la metodología a la realidad organizacional existente. El Coordinador decidirá qué roles, artefactos, y prácticas serán utilizados y planificará un proyecto piloto en donde se los pondrá a prueba. Este proyecto será elegido del conjunto de proyectos reales planteados en el negocio de forma que se pueda evaluar la aplicación de AgEnD sobre casos concretos.

Adaptación de las Personas (AP)

Para analizar cómo las personas interactúan dentro de una organización y lograr la institucionalización del proceso, AgEnD particiona el análisis de esta disciplina en tres perspectivas. La primera de ellas, es la perspectiva organizacional o ambiental que describe la cultura organizacional y los objetivos al adoptar una metodología ágil. La segunda perspectiva, la perspectiva del equipo de desarrollo, describe como los miembros del equipo interactúan entre sí y con el ambiente que lo rodea. Por último, la perspectiva del individuo está relacionada con la psicología y la personalidad de cada individuo y como esto toma relevancia al momento de seleccionar el rol adecuado según el perfil.

Las disciplinas de AP y AM presentan el workflow mostrado en la Figura 035 para el análisis de la organización al momento de la implementación. En dicha figura se detalla como las dos actividades principales de la Implementación de AgEnD ocurren

en paralelo y en forma iterativa, hasta que todos los cambios planificados han sido ejecutados y existe concurrencia respecto al éxito de la adopción de la metodología.

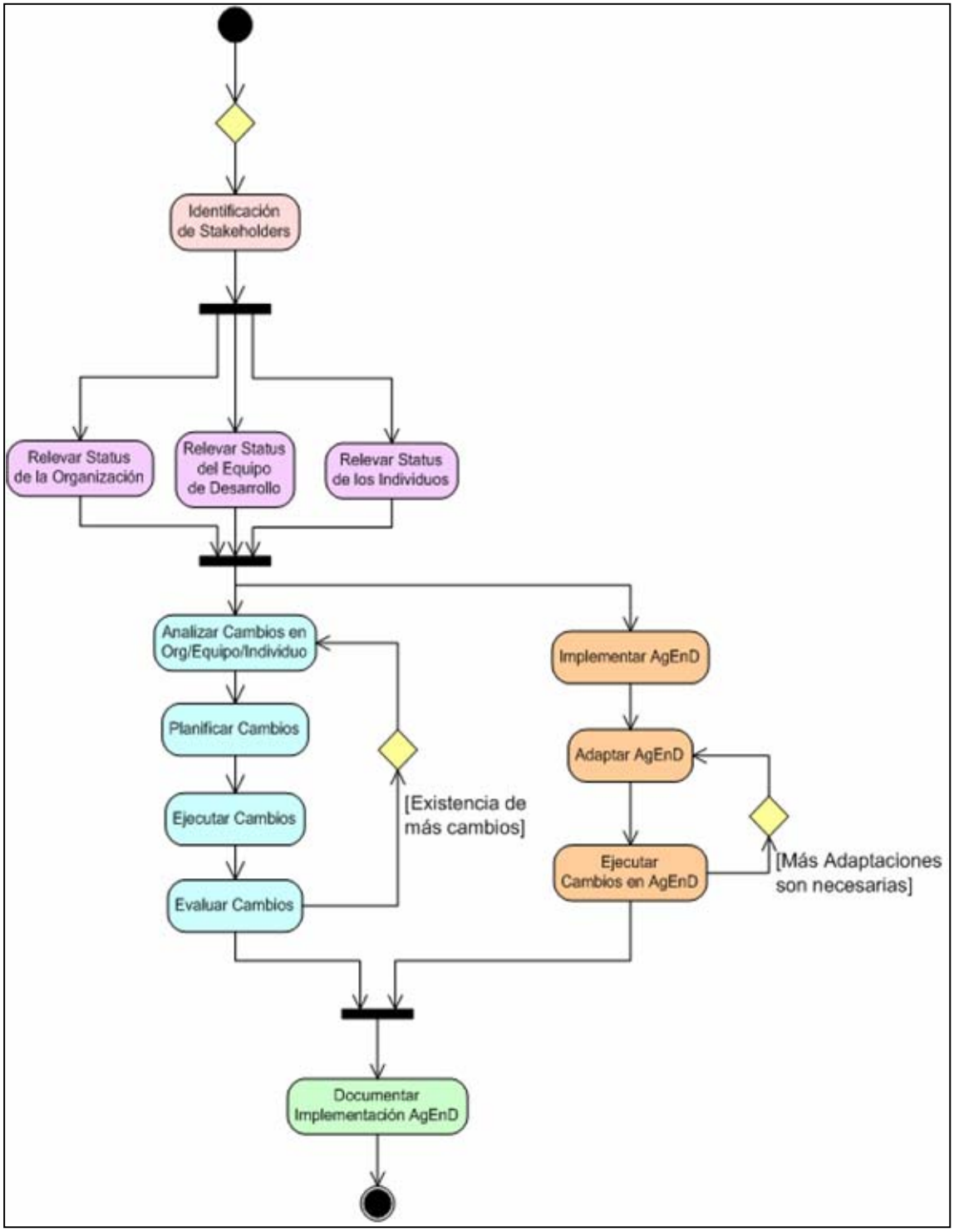

Figura 035. Diagrama de Actividad de la implementación de AgEnD.

A continuación dividiremos el análisis en los tres factores que se muestran el la Figura 035: Organización, Área de Desarrollo, Individuo.

La Organización

Una organización puede ser observada como un sistema abierto. Existen inputs, un proceso de transformación que genera outputs, y un ciclo de feedback que ayuda a controla que los outputs sean los esperados para poder tomar acciones correctivas. Sin embargo, pensar que este modelo alcanza para describir las actividades fundamentales de una organización es erróneo ya que la organización reviste un comportamiento dinámico resultante de una multiplicidad de factores que confluyen para conformar la identidad organizacional los cuales no pueden ser tomados linealmente – sobre esto se podrá consultar [Etkin, 1989].

En el inicio de la implementación de AgEnD se efectúa un relevamiento del status organizacional. Para esto se deberán identificar aquellos stakeholders claves para que el proceso de implementación resulte exitoso. En la Tabla 001 se describen aquellos stakeholders que resultan importantes incluir al momento de llevar a cabo el relevamiento.

Stakeholder	Influencia
Sponsor Ejecutivo	Persona responsable de obtener el soporte ejecutivo necesario para el éxito de la implementación de una metodología ágil
Gerente de Sistemas	Persona a cargo del área de sistemas que administra el sector
Project Manager	Persona/s a cargo que coordinan los proyectos de desarrollo
Líder de QA	Persona a cargo del área de QA (Quality Assurance) en la organización
Coordinador	Persona a cargo de la implementación de la metodología en la organización

Tabla 001. Stakeholders involucrados en el proceso de implementación de AgEnD.

El mismo será realizado mediante la realización de un Workshop de Status Organizacional (WSO) en el cual se contará con la participación de los stakeholders que tendrán la mayor influencia en la implementación de AgEnD. El objetivo de este workshop es el de poder formalizar ciertos aspectos que darán comienzo al proceso de

implementación y de lograr concurrencia en relación a los objetivos organizacionales de mejora del proceso. Como resultado el mismo debería dar:

- Especificación de los objetivos organizacionales de la implementación
- Especificación de cómo el proceso influenciará a sus usuarios y demás stakeholders
- Status del proceso actual y las principales razones que influyeron en el cambio
- Asesoramiento de la *cultura organizacional* para poder planificar cualquier intervención para que las personas adapten su forma de trabajo

El workshop permite entender cómo la organización manejará la adopción de un nuevo proceso de desarrollo de software de forma exitosa. En el caso de AgEnD, o de cualquier otra metodología ágil a ser implementada en una organización, es importante considerar que se requiere una cultura abierta en la que se fomente la comunicación e interacción de los individuos (tema a ser tratado en las etapas de AP). El resultado de este workshop será plasmado en algún medio de información de la empresa para que las personas involucradas puedan conocer el avance en la implementación.

Según los estudios de otros investigadores citados por Jim Highmith [Highsmith, 2002] en relación a la implementación de metodologías ágiles, existen cuatro tipos básicos de culturas organizacionales; cada una está caracterizada por una motivación que permite que la misma se mantenga en forma invariable. Las cuatro culturas son: la cultura de competencia, la cultura de colaboración, la cultura de control y la cultura de cultivo.

La cultura de competencia enfatiza la capacidad de los individuos, su responsabilidad y su conocimiento. De alguna forma los niveles jerárquicos son dados por aquellas personas que más conocimiento tienen. En éstas los títulos son algo secundario y las cuestiones políticas quedan de lado.

La cultura de colaboración se da cuando se tienen equipos multidisciplinarios. El liderazgo se establece en relación al rol en vez del título en sí. Generan una importante cantidad de innovación dando una gran relevancia a la formación de equipos internos para las distintas áreas.

La cultura de control está basada en el poder y las grandes jerarquías. En las mismas se tienen mecanismos políticos muy fuertes lo que las hace a veces bastante burocráticas. Sin embargo, dado su tamaño, se mantienen en el mercado por largo tiempo. La prioridad en ellas es la planificación de las tareas, y el seguimiento y control de las actividades realizadas por todos los recursos.

La cultura de cultivo es similar a la cultura de colaboración con la diferencia que se prioriza a los individuos por sobre los equipos. Idealmente tienen su nicho en aquellas empresas que realizan productos nuevos o grandes innovaciones. En general son auto-organizativas y carecen de formalismos en cuanto estructura, procesos o documentación.

Analizando estas nociones se identifica a AgEnD como una metodología que tendrá mayores éxitos si es implementada en culturas de competencia y de colaboración. Las razones de esto tienen que ver con que ambas priorizan a la gente por sobre los procesos y los rigurosos controles sin por eso caer en el caos o la auto-organización de las personas (algo que sí ocurre en la cultura de cultivo). Si en el workshop antes mencionado se resuelve que la cultura de la empresa tiene similitud con alguna de estas dos, las probabilidades de una implementación de exitosa AgEnD dentro de la organización serán altas.

La cultura de control requiere niveles metodológicos de mayor peso (lo que a veces puede caer en una extremada burocracia) que le permitan mantener las jerarquías de poder y la planificación continua de los proyectos. Por estas razones y por poner el énfasis en la estructura y la organización por sobre las personas, las metodologías ágiles no serán una buena opción. En estos casos se podrán realizar proyectos pilotos de implementación de AgEnD pero la metodología en cuestión rara vez será institucionalizada.

Finalmente, como ya fue mencionado la cultura de cultivo rechaza en cierta forma cualquier tipo de organización más la propia generada por las personas. Además mantiene a los individuos y su continuo deseo de hacer lo que más les interese, por encima de la colaboración de grupos de trabajo. Esto hace que el peso inherente a cualquier metodología (inclusive AgEnD) sea intolerable y probablemente al poco tiempo la iniciativa sea dejada de lado.

Si se intenta algún tipo de cambio organizacional ya que la cultura resulta inadecuada para los objetivos de la empresa, el mismo deberá ser realizado cuidadosamente, teniendo en cuenta la resistencia al cambio mencionada anteriormente. Como ya fue explicado existen métodos de intervención organizacional que pueden ayudar durante la mejora de proceso [Wainstein, 2000].

El Área de Desarrollo

Sin duda alguna el área que estará sometida a la mayor influencia será el Área de Desarrollo. Aquí se encontrarán los "usuarios finales" del proceso que serán los que participarán en la customización del mismo. Para ejecutar la implementación se evaluará primero el enfoque de la **Adaptación Metodológica** en que cada disciplina, artefacto y actividad de AgEnD será contrastada con la forma de trabajo de las personas para poder elegir la mejor manera en que estás últimas podrán absorber el proceso.

Teniendo en cuenta las diferentes áreas de conocimiento de la Ingeniería de Software, propuestas en el SWEBOK [SWEBOK, 2002], identificaremos los roles de cada individuo en relación a las mismas. Las áreas en cuestión son: Requerimientos de Software (SR), Diseño de Software (SD), Construcción de Software (SC), Testing de Software (ST), Mantenimiento de Software (SM), Administración de la Configuración del Software (SCM), Administración de Ingeniería de Software (SEM), Proceso de Ingeniería de Software (SEP), Métodos y Herramientas de Ingeniería de Software (SETM), Calidad de Software (SQ).

Cada una de estas áreas puede estar o no contemplada dentro de la organización y ser realizada por uno o más roles. La realidad del mercado informático en consultoras de tamaño modesto en la Argentina indica que es común que la estructura esté formada por un grupo importante de personas en el área de SD/SC, un pequeño grupo de personas en SR/ST, grupos de personas dispersas en SM, y un mínimo número, o ninguno en algunos casos, de personas abocadas a SEP/SETM/SQ. Es decir, aquellas actividades cuyo ROI es más inmediato concentran la mayor cantidad de recursos en detrimento de aquellas actividades cuyos beneficios solo son visibles a largo plazo, que tienden a contar con poco personal.

A consecuencia de esta realidad planteada se deberá incluir al menos un recurso responsable de la implementación del proceso. Como sugiere AgEnD esta persona tendrá el rol de *Coordinador* y supervisará el proceso velando por que el mismo sea implementado correctamente pudiendo customizarlo de acuerdo a las necesidades de sus usuarios.

Regresando a las áreas de conocimiento de la Ingeniería de Software antes mencionadas no todas demandan el mismo grado de *Adaptación Metodológica*. Dicho grado estará dado principalmente por los requerimientos comunicacionales y el nivel de predictibilidad de los roles asociados a dicha área de conocimiento. Como se observa en la Figura 036, a medida que abarcamos áreas más predecibles las tareas de customización dadas por la *AM* disminuyen. Por ejemplo, las actividades relacionadas con SCM suelen estar más estandarizadas y presentan una menor variabilidad; esto se traduce en un esfuerzo menor al momento de adoptar las prácticas. Especialmente esto es cierto en aquellas área de conocimiento que han logrado una importante automatización por sobre aquellas que todavía son realizadas de manera más "artesanal".

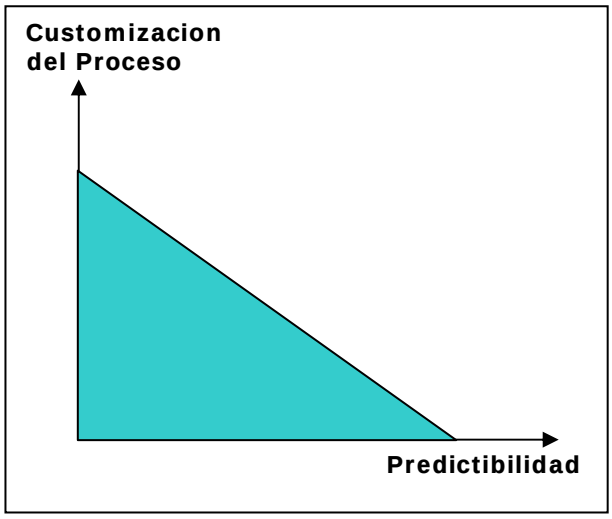

Figure 036. Customization vs. Predictibilidad del Rol.

La Figura 036 manifiesta la necesidad de customizar aquellas áreas que poseen una mayor preponderancia del factor humano y las cuales se encuentran en un escalón de madurez más bajo dentro de nuestra ingeniería. El Coordinador comenzará mediante el análisis de cómo cada área de conocimiento será implementada dentro del área de desarrollo de la organización. Será relevante presentar las técnicas que puede utilizar el

Coordinador para las actividades de Adaptación de las Personas. Alistair Cockburn [Cockburn, 2001a], quien estudió el desempeño de los grupos de desarrollo de software, sugiere cuatro herramientas sencillas que pueden lograr que las personas que participan del desarrollo se aliñen para entender los beneficios de la mejora de proceso. Éstas consisten en darle a la gente:

- Información adicional respecto de los objetivos de los proyectos
- Información adicional acerca del efecto de sus acciones, para que identifiquen aquellas acciones que van en contra de los objetivos
- Una mejor razón para que empujen en la dirección deseada

Como se observa, muchas de estas cuestiones parten del sentido común pero pueden servir para que un equipo de desarrollo que presenta cierta resistencia a AgEnD comience a alinear sus esfuerzos hacia la implementación de las prácticas y la articulación de los principios. Podemos mencionar que la disciplina de Administración de Proyecto dentro de AgEnD debe aportar toda la información necesaria para que el equipo sepa hacia dónde dirigir sus esfuerzos.

El Individuo

El desarrollo de software es una actividad humano-intensiva. Dada la complejidad inherente al software las personas se ven obligadas a trabajar en forma grupal interactuando con sus pares. En estas interacciones que ocurren a diario en las organizaciones se juegan “juegos de poder”. Estos determinan el comportamiento de las personas las cuales ejercen su autoridad y son sujetas a las autoridades de otros.

Si recurrimos al Modelo Motivacional de la Figura 037 propuesto por Harold Leavitt, observamos que las personas están continuamente buscando el equilibrio interno; cuando algo desafía este estado de equilibrio, existe un sentimiento de frustración que genera alguna acción en la persona y que hace que esta cambie buscando un nuevo estado de satisfacción. De hecho, este ciclo continúa en todos los ámbitos de la vida humana durante la existencia del individuo y es de utilidad entenderlo ya que la satisfacción/motivación de las personas en su trabajo determinará ulteriormente el resultado de la implementación de un proceso ágil de software.

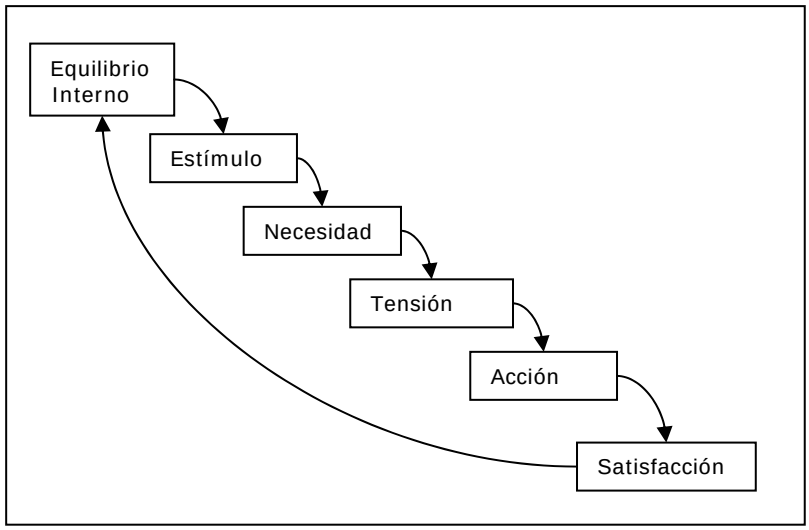

Figure 037. Modelo Motivacional para una persona.

Desde un punto de management, acorde a las palabras de Tom Peters, una de las consideraciones principales que tendrán las empresas del mañana será que su valor de mercado estará dado por los bienes intangibles que esta posee (a diferencia de la visión tradicional en que se valoran principalmente los tangibles). Los bienes intangibles están dados por el intelecto e imaginación de las personas – denominadas *knowledge workers* bajo este esquema.

Como ya fue descrito el proceso plantea ciertos principios, prácticas y patrones que conviene sean utilizados para lograr la satisfacción continua de la gente que lleva a cabo la construcción del software. La satisfacción de cada uno de los individuos contribuirá a la motivación del equipo y ayudará en la implementación del proceso.

Dentro del espectro de comportamientos humanos, una metodología tiene dos opciones para lidiar con la diversidad de personalidades: ser tolerante a la individualidad o ser estricta y disciplinada para lograr comportamientos determinados. Dado que cada individuo es único y posee una personalidad que lo caracteriza, AgEnD plantea un marco de implementación tolerante a la diversidad de personalidades y formas de trabajo de los individuos. Para que la tolerancia de la metodología tenga éxito y la construcción del software no sea algo caótico se requieren ambientes de colaboración, con buenas relaciones interpersonales, y un cierto grado de empatía por hacer el trabajo y ayudar a otros a que lo hagan. En general podemos decir que a veces la disciplina resulta conveniente y aunque sea difícil de lograr puede resultar más

eficiente. La tolerancia puede resultar más fácil de adoptar pero quizás sea menos productiva.

Aportes de AgEnD al Espectro Metodológico

A continuación se expondrán cuales son los aportes que realiza AgEnD al universo de las metodologías ágiles, en base a un análisis exhaustivo de los procesos existentes y de la identificación de aquellos aportes que hacen de AgEnD un proceso ágil único.

Nomenclatura Estandarizada

Una de las primeras facilidades que una persona se encuentra al aprender AgEnD es el uso de nombres estandarizados para todos sus elementos. Muchos de los nombres utilizados han sido tomados del RUP [RUP, 2002] que sirve como base de conocimiento para procesos de desarrollo. También existieron fuertes influencias del SWEBOK [SWEBOK, 2001] que comprende el cuerpo de conocimiento de la ingeniería de software y que es fiel esperanza del autor que dicho proyecto siente las bases hacia una profesionalización de nuestra disciplina.

Gracias a esta estandarización, AgEnD contribuye a un global entendimiento de los conceptos involucrados lo cual habilita una comunicación más fluida entre las personas involucradas.

Administración de Proceso

AgEnD propone distintos patrones enmarcados en una disciplina que pueden ser utilizados para customizar el proceso en distintas etapas. Los patrones sirven para que AgEnD pueda ser implementado exitosamente y para adaptar el mismo a las necesidades particulares del proyecto. Esto logra captar el feedback de la gente que es usuaria del proceso y dinámicamente modificarlo para que cumpla mejor sus objetivos.

Administración de Personas

AgEnD propone distintos patrones enmarcados en una disciplina que pueden ser utilizados para manejar mejor el factor humano dentro del desarrollo. Los patrones ayudan a mejorar el ambiente de trabajo de los individuos, mejorar la motivación de los

grupos de trabajo y mitigar las fricciones que puedan existir. El resultado será una adopción rápida y eficaz de AgEnD, tolerante a la diversidad de personalidades existentes.

Administración del Conocimiento

AgEnD explícitamente establece una disciplina de Administración del Conocimiento que contribuye al aprendizaje organizacional. La misma establece una serie de prácticas ágiles que ayudan a que el conocimiento novedoso generado durante un proyecto sea capturado para una posterior recuperación. El objetivo es que el conocimiento se disemine por los grupos de trabajo y que se vaya aprendiendo de la experiencia para el mejoramiento de las personas y la agilización del proceso.

Procedimiento para la Implementación

Existe una frase en el terreno de la Ingeniería de Software que dice: “Lo difícil no es definir una metodología, sino implementarla en la organización”. Dada esta realidad, AgEnD plantea una serie de consideraciones, surgidas a partir de las experiencias de otras áreas de conocimiento, que contribuyen a su implementación en un grupo humano.

Prácticas Probadas por la Experiencia

Las prácticas propuestos por AgEnD no son invenciones del autor ni conceptos teóricos ultra novedosos. Son prácticas que han sido probadas durante suficiente tiempo como para concederles un status de “best practices”. Las mismas han sido estudiadas en libros mencionados en la bibliografía y se reconoce su valor agregado para mejorar el proceso de construcción de software.

En especial, AgEnD reúne un conjunto de prácticas que considera adecuadas a la aplicabilidad de la metodología y que ha sido demostrada por la experiencia propia del autor en diversos proyectos (ver más información en el capítulo de Resultados Experimentales).

Templates de Artefactos

AgEnD ofrece una serie de templates (ver Apéndice A) que ayudarán al Coordinador que lleve a cabo la implementación a elaborar los artefactos sugeridos. Estos han sido utilizados en distintos proyectos y plasman de una manera concisa la información que se desea transmitir. La idea de los mismos es adaptarlos de acuerdo a la realidad de los proyectos manejados por la organización.

Baja Curva de Aprendizaje

AgEnD está pensado para ser utilizada tal cual establece esta tesis. No es necesario poseer conocimiento avanzados en conceptos teóricos ni disponer de una estructura organizacional dedicada exclusivamente a la implementación del proceso. Las prácticas han sido elegidas entre otras cosas por adaptarse a la forma de trabajo de la gente ayudando a resaltar aquellos elementos que hacen a una metodología ágil.

Asimismo las prácticas resultan sencillas de transmitir y de implementar para toda persona que conozca las ideas detrás de los procesos de desarrollos iterativos e incrementales.

Capitulo IV - Resultados Experimentales de las Prácticas de AgEnD

A continuación se describirán dos experiencias de desarrollo en las que se utilizó AgEnD como metodología. En ambas el autor formó parte del equipo de desarrollo teniendo a su cargo el análisis de la realidad organizacional previa, la adaptación inicial de la metodología, la capacitación a las personas, y la customización dinámica en intervalos regulares que permitiera adecuar AgEnD al proyecto en cuestión.

Las experiencias muestran como AgEnD contribuyó a que los proyectos lleguen a su concreción exitosamente – sin por eso restar importancia a los equipos de trabajo que son el factor más importante del desarrollo. Describiremos cada proyecto con un nombre clave por cuestiones de confidencialidad. Se ese dará un nombre de fantasía a cada proyecto, el primero será FIDoNET y el segundo será CONEST.

Experimentación de Prácticas utilizando AgEnD

Primer Ejemplo: Aplicación en FIDoNET

A continuación, veremos como las diversas prácticas recomendadas por AgEnD han sido probadas experimentalmente tanto por el autor de esta tesis como por diversas personas que han diseñado reconocidas Metodologías Ágiles.

Plantearemos la experiencia de un proyecto que es encarado por una empresa que presta servicios a ONGs, la cual es llamada FIDoNET[9] que realiza desarrollos a medida para clientes de gran envergadura. FIDoNET ha decidido crear un sistema para la administración de las actividades que posee un ONG y la interacción con sus usuarios. Dicho sistema será utilizado en distintos países de Latinoamérica en que FIDoNET tiene contactos y clientes, por lo que deberá ser multilenguaje.

Background del Proyecto

El proyecto en cuestión tiene como objetivo construir una herramienta que brinde servicios a las ONG y a las personas que pertenecen a estas. El proyecto es conocido internamente como FIDoNET y el lema que impulsa al sistema es el siguiente:

> *"Construyamos relaciones solidarias donde los deseos y las actividades en común se constituyan en la sociedad mediante una herramienta humanista, que se adecue a las singularidades."*

FIDoNET ha decidido utilizar un equipo de desarrollo que inicialmente será de 4 personas, y posteriormente en la construcción crecerá a unas 5 personas. El proyecto será conducido utilizando AgEnD como proceso, siendo este proyecto la prueba piloto de la implementación y utilización de la misma. Este es el primer proyecto de gran envergadura en que la empresa utilizará esta metodología. Inicialmente, AgEnD será utilizado en forma completa sin realizarse ninguna modificación; a medida se avance en las iteraciones se irán realizando reuniones de post mortem para configurarlo acorde a las necesidades del equipo de desarrollo. Todo el equipo detrás del emprendimiento de

[9] Por cuestiones de confidencialidad se dará un nombre de fantasía de la empresa.

FIDoNET está comprometido a utilizar AgEnD en todas sus fases pudiendo adaptarlo a medida se avanza en el desarrollo mediante la intervención del Coordinador.

Inicialmente los stakeholders del proyecto forman un grupo de considerable diversidad. En el mismo incluimos a ONGs, Empresas asociadas con FIDoNET, Instituciones, Usuarios individuales y recursos de FIDoNET. El gran porcentaje de comunicación en el desarrollo será realizado con dos usuarios expertos de la propia empresa los cuales conocen el sistema a construir en detalle. Ellos recibirán pedidos de los demás stakeholders y los canalizarán en el proyecto al fin de cada iteración.

Roles y Recursos

Los roles de AgEnD están llevados a cabo por distintas personas del proyecto. A continuación mencionamos a los trabajadores que participaron en la primera versión del proyecto:

- Existían dos *Usuarios Expertos*, quienes estuvieron en contacto con las ONGs siendo el input funcional para el *Equipo de Desarrollo*. A lo largo de este capítulo nos referimos a estos como “el Cliente”.

  Reseña: Los Usuarios Expertos conformaban un dúo cohesivo que posee características que los hicieron ideales para dichos roles. Eran personas accesibles, informales, flexibles y tolerantes. No trataban de imponer sus decisiones sino que permitían espacios de discusión en cualquier ámbito. Tenían muchos contactos en la comunidad de ONGs de la Argentina.

- Existía un *Patrocinante* o *Sponsor* que veló por mantener el proyecto en su rumbo, eliminando obstáculos que estuviesen fuera del alcance del *Equipo de Desarrollo.*

  Reseña: El Sponsor fue la persona que guío el emprendimiento FIDoNET. Sus aptitudes eran las de ser un muy buen negociador, poseer contactos con diversas áreas del medio, entender el proyecto en forma íntegra. Era el encargado de verificar el progreso de las distintas áreas y de financiar el proyecto.

- Existía un *Líder de Proyecto* que cumplió también funciones de *Coordinador de Proceso* y de *Administrador de Conocimiento.*

  Reseña: El Líder de Proyecto fue la persona encargada de gestionar el proyecto. Asimismo era entusiasta de las metodologías ágiles, proponiendo la utilización de AgEnD como proceso de desarrollo del proyecto.

- Existía un *Analista* que también cumplió funciones de *Tester*.

  Reseña: El Analista era una persona con mucha habilidad y experiencia en las cuestiones humanas, el trato con el cliente, técnicas de relevamiento y especificación de requerimientos. La técnica de Casos de Uso fue relativamente nueva para esta persona por lo se requirió alguna capacitación.

- Existía un *Arquitecto* que también cumplió funciones de *Desarrollador*.

  Reseña: El Arquitecto era una persona con mucha habilidad y experiencia en las cuestiones técnicas del desarrollo. Llevaba años trabajando en tecnologías Java y trabajó como arquitecto en proyectos de gran escala.

- Existían dos *Desarrolladores* que poseían distintos grados de experiencia en las tecnologías del proyecto.

  Reseña: Los dos tenían cargo de Programador Senior y conocían a fondo la tecnología seleccionada. Conocíann la Plataforma Java 2 en forma teórica y práctica ya que poseen abundante experiencia laboral.

Tecnologías

Dadas los objetivos globales de FIDoNET como organización de crear unión entre personas fomentando la solidaridad, se priorizó al momento de selección de tecnologías aquellas que respondan al modelo de desarrollo Open Source y que puedan usar libremente sin costos de licencias asociados. Conjuntamente, otro de los factores a

tener en cuenta en la selección es la madurez de la herramienta/tecnología/framework en la industria, descartándose aquellas que estuvieran en sus fases iniciales de desarrollo.

Arquitectura de la Aplicación

Arquitectura Web: presenta la interfase al usuario mediante un navegador de internet pudiendo desacoplar las capas de interfase/lógica de negocio/datos en el servidor mediante frameworks disponibles.

La arquitectura de la aplicación para prestar servicios web estuvo compuesta por las siguientes capas (tiers):

- Capa de Presentación: páginas web servidas mediante un framework de MVC que desacopla esta capa de las capas de lógica del negocio y manejo de datos. Asimismo, se tuvo especial énfasis en utilizar un medio que permita que se muestre esta capa en distintos dispositivos (Ej.: PDA, celulares, tablet-pc, sistemas embebidos, etc.) sin tener que hacer cambios importantes a la aplicación.
- Capa de Lógica del Negocio: inicialmente, se pensó en un esquema sencillo de lógica del negocio que simplemente recibe/envía la información a la capa de presentación, mediante la interacción directa con la capa de manejo de datos. Se planteó un esquema de componentes que resuelvan los pedidos realizados sin necesidad de utilizar un application server. Sin embargo, si en algún momento se veía la necesidad de que la aplicación escale y que se pueda adaptar a requerimientos no funcionales más fácilmente, se vería la utilización de un application server el cual resuelva cuestiones como persistencia, seguridad, manejo de transacciones, logging, etc.
- Capa de Manejo de Datos: para esto se utilizó un motor de base de datos relacional. El mismo es accedido mediante un esquema de persistencia que hace transparente para la aplicación el mapeo objetos-relacional (ORM). Para ello, se analizaron diversos frameworks que resuelven estas cuestiones.

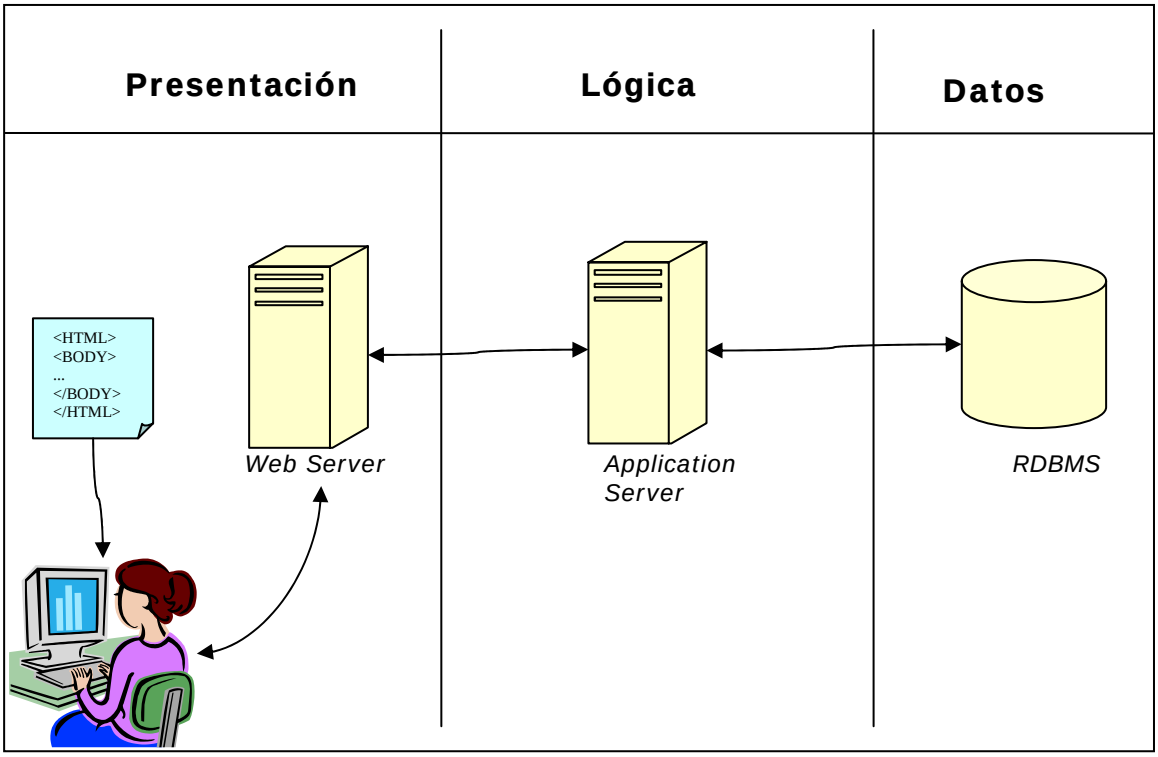

Figura 037. Diagrama de nodos físicos de la aplicación web de FIDoNET

Plataforma Base y Herramientas

Linux: inicialmente se plantea la utilización de este SO como servidor que contenga la lógica del negocio y el motor de BD

Microsoft Windows: SO utilizado en las máquinas de desarrollo

Apache Tomcat: Web Container

CVS: Sistema de Administración de la Configuración

Eclipse: IDE de Desarrollo

Macromedia Dreamweaver: Herramienta de diseño de páginas web

Motores de BD

MySQL: uno de los motores Open Source más populares, MySQL es utilizado en la mayoría de los sitios web del mundo en forma conjunta con el lenguaje de scripting PHP. El motor es muy robusto y si bien no posee toda la funcionalidad de un motor como Oracle, es una opción más que interesante por la rapidez y sencillez del mismo.

Framework de MVC

Jakarta Struts: framework utilizado para la capa de presentación de la aplicación. El mismo se utiliza para armar el patrón MVC pudiendo desacoplar la presentación de la lógica del negocio.

En particular, Struts utiliza JSP para enviar la página al cliente que efectúa el request (View Component). En el mismo se podrán usar tags propios de la librería de Struts que permiten minimizar el código que se escribe en el JSP.

Struts ha madurado y es utilizado en todo el mundo. La comunidad de usuario de Struts crece a un ritmo considerable. Existen muchas aplicaciones complementarias disponibles para usar con el framework y muchos desarrolladores haciendo contribuciones al mismo.

Object Relational Mapping (ORM)

Torque – Apache DB Project: framework que se utiliza para realizar la capa de persistencia de la aplicación. El mismo provee una forma de acceder al motor de RDBMS sin necesidad de escribir código SQL dentro de la aplicación.

De esta forma permite abstraer la persistencia facilitando el mantenimiento del código ante cambios en la representación de los datos o en la tecnología de BD utilizada.

El mismo es utilizado ampliamente por la comunidad Open Source, y ha madurado hasta consolidarse como un importante y estable framework que actualmente se encuentra en su versión 3.0.2.

Lanzamiento del Proyecto

FIDoNET comienza el proyecto en la fase denominada Factibilidad, durante la cual se genera un modelado del negocio sobre el cual será construido el sistema. El propósito es entender toda la problemática que tienen las ONGs en la Argentina para

poder acotar el campo de funcionalidad que será incluido en la aplicación. Esta función es llevada a cabo por las personas Responsables del Negocio que conocen en profundidad a los stakeholders del futuro sistema y los servicios que serán provistos a estos. Durante un período de 4 meses estas personas mantienen reuniones periódicas en las que se identifican aquellas ONGs que servirán como referentes al momento de relevar necesidades.

Se planifican las fases en que el proyecto de FIDoNET será dividido tomando como referencia las fases de AgEnD. Se realiza una estimación inicial de muy alta granularidad para que los stakeholders vayan teniendo visibilidad y también para asegurar la viabilidad económica del proyecto.

Resultados del Proyecto

Después de seis meses de desarrollo el proyecto llega a un hito importante, la entrega del release 1.0 que concluye la puesta en práctica de AgEnD. El mismo ha terminado en forma satisfactoria para las partes involucradas. Se han implementado todas las prácticas recomendadas por AgEnD y se realizó una reunión de Cierre de esta primer etapa para relevar y analizar el proyecto y la metodología. A continuación se comentan aspectos de la metodología que contribuyeron al éxito del proyecto.

Configuración del Proceso

AgEnD recomienda ir customizando la metodología de acuerdo a las necesidades del proyecto y de las personas. Se llevaron a cabo varias reuniones de revisión de la metodología en la que se fueron ajustando diversos aspectos de la misma según se veía la forma de trabajo de la gente y la naturaleza del proyecto.

En el proyecto de FIDoNET una de las primeras cuestiones que se observó era que los Casos de Prueba insumían mucho trabajo para el Tester el cual en general tenía que estar bastante tiempo realizándolos. Este optó por anotar en una planilla de cálculo un breve mensaje dejando registrado el nombre del caso de prueba y por cada fila el resultado de la ejecución en una versión dada de la aplicación. Esto aceleró sus tareas sin degradar el testing efectuado.

Algo similar ocurrió con los Casos de Uso que inicialmente fueron basados en el template de [RUP, 2002]. Los mismos eran especificados por el Analista en conjunto con los Usuarios Expertos. Durante las primeras iteraciones se observó que los Desarrolladores los leían por arriba y hacían las preguntas directamente a los Usuarios Expertos quienes hacían un breve repaso por el flujo de interacciones. A partir de ese momento se decidió especificar el comportamiento en casos de uso mucho más informales que permitieran que los Desarrolladores tuvieran un idea de lo que tenían que implementar pudiendo profundizar con la comunicación cara a cara con los Usuarios Expertos. Finalmente, los casos de uso terminaron siendo bastante parecidos a las *User Stories* planteadas por XP [Jeffries, 2001].

Tomando el concepto del *Scrum Daily Meeting* sugerido por Scrum [Schwaber, 2001], se decidió agregar dentro de la disciplina de Administración de Proyecto de AgEnD esta reunión diaria de quince minutos a primera hora de la mañana con el equipo de desarrollo. La misma era coordinada por el Líder de Proyecto quien preguntaba a cada persona del equipo qué había hecho de la durante el día anterior, que iba a hacer durante ese día y que obstáculos tenía para llevar a cabo sus actividades. En la reunión podía participar el Cliente pero solo como observador. Esto permitió ir teniendo un control y monitoreo constante sobre el progreso y los riesgos que se iban dando.

En relación a los artefactos sugeridos por AgEnD se dejaron de lado la Lista de Riesgos ya que con la reunión diaria del Líder con el equipo el control y monitoreo de los mismos era on-line. También no se vio la necesidad de generar Notas de Entrega ya que el equipo de desarrollo era parte del Cliente y existía plena confianza e involucramiento de este último en el proyecto.

Administración de la Configuración

Dentro del contexto de un proceso moderno de desarrollo es indispensable contar con una herramienta que automatice todas las tareas relacionadas con los ítems de configuración generados durante un proyecto. Para esto se seleccionó la herramienta CVS (Concurrent Versions System) que permite llevar esta tarea a cabo.

De acuerdo a lo recomendado por AgEnD todos los artefactos generados durante un proyecto serán puestos bajo control de versiones. Para comodidad del equipo de

desarrollo se decidió tener dos módulos independientes dentro de un mismo repositorio. El repositorio tenía el nombre del proyecto y existían dos módulos: uno denominado *Project* y otro *Sources*. El primer módulo incluía toda la documentación asociada a las distintas disciplinas de AgEnD categorizadas en directorios con el nombre de la disciplina. El segundo módulo tenía todo el código de la aplicación a ser cargado en la IDE utilizada – el Eclipse.

Esta disciplina permitió que en un momento determinado se pudieran probar ciertas implementaciones de casos de uso utilizando un branch separado manteniendo consistente la rama de desarrollo principal en la que se iban liberando releases al Cliente.

Máxima Comunicación

Una de las prácticas que fue aplicada con mayor éxito en el proyecto fue la de Máxima Comunicación. Todas las personas involucradas en el proyecto (tanto de parte de los usuarios expertos como el equipo de desarrollo) estaban en un mismo piso de un edificio en oficinas adyacentes. Esto permitía que cualquier duda planteada por cualquier parte se resolviera mediante la rápida consulta con el miembro que tenía el conocimiento. Esta es una práctica esencial en AgEnD que es prerrequisito de su implementación. Alistair Cockburn [Cockburn, 2001a] identifica como las corrientes comunicacionales mejoran la transferencia de conocimiento y son una de las causas por las cuales se pueden minimizar el overhead metodológico. Esto fue tratado ampliamente en el capítulo anterior.

En caso que esta práctica no se hubiera implementado no se podría haber implementado una metodología como AgEnD ya que el grado de ceremonia de esta no es conforme a ambientes en donde se debe suplir la carencia de comunicación mediante comunicación. Para esto se recomienda usar alguna metodología como el Unified Process.

Participación Activa del Cliente

De acuerdo a la práctica de Máxima Comunicación el Cliente estuvo involucrado durante todo el proyecto. El mismo participaba en las reuniones diarias de

avance solo como observador y en las reuniones semanales de revisión para validar el contenido de lo que se construyó.

El Cliente junto con el Analista tenía a su cargo la especificación de los casos de uso. Al final la aplicación contaba con 42 casos organizados en distintos módulos que mapeaban con el formato de la interfase gráfica. Otra de las prácticas importantes que se utilizó en relación al Cliente fue un Control de Cambios formal. Para esto el Cliente dejaba en el Gantt del Proyecto distintos pedidos de cambio. Cada uno de estos era evaluado por el Líder de Proyecto y por el Arquitecto quienes discutían el impacto del cambio y el costo en recursos/tiempo del mismo. Las posibilidades eran hacer el cambio en la próxima iteración, dejarlo para la próxima versión del sistema o desecharlo completamente. Esto funcionaba como escudo para los Desarrolladores quienes de otra forma hubieran tenido que estar continuamente cambiando el código que desarrollaban dado el ambiente de alta volatilidad existente.

Estimaciones Ágiles

Esta práctica fue utilizada frecuentemente al principio del proyecto y luego continuadamente en cada iteración. Durante la primera fase se tomaron los features del documento de Visión del producto los cuales comenzaron a ser pasados a casos de uso. El Líder de Proyecto en forma conjunta con el Arquitecto y el Cliente, dieron prioridad a estos casos de uso seleccionando la versión del producto en que cada uno aparecería. De acuerdo a la propuesta de AgEnD se tomaron los casos de uso seleccionados para la primera versión de manera de efectuar una estimación global de la duración del proyecto. Se dejó en claro que dicha estimación arrojaría un resultado aproximado y dada la naturaleza de los procesos ágiles el mismo simplemente serviría para planificar hitos importantes del proyecto.

Usando la metodología de UCP (Use Case Points) junto con los factores de ajuste correspondientes, la estimación arrojó un total de 215 UCP no ajustados. Mediante la utilización del factor de ajuste estandar de la industria de 20 horas/persona por UCP se obtuvo un esfuerzo para el proyecto de 18,78 meses/persona. De acuerdo al equipo de desarrollo descrito anteriormente se estimó la duración en 5 meses.

Este resultado fue bastante acertado si se tiene en cuenta que la primer versión fue liberada 6 meses después. Cabe remarcar que durante el desarrollo hubo grandes

cambios en los requerimientos y la estructura y priorización de los casos de uso se modificó considerablemente.

Enfoque en la Arquitectura

Esta práctica fue de gran ayuda en el desarrollo ya que permitió definir cuestiones técnicas que impactaría positivamente en la solución al problema. Una vez que se comenzó con la especificación de los casos de uso, en paralelo el Arquitecto junto con los Desarrolladores comenzaron a definir la arquitectura del sistema. Uno de los requerimientos no funcionales más importante era el construir un sistema mantenible y flexible, el cual pudiera incorporar cambios con facilidad. Existía también una fuerte restricción de diseño de mantener los costos lo más bajo posibles y de que el sistema no tuviera licenciamiento alguno por lo cual se debían utilizar componentes Open Source.

Tomando como input los casos de uso que se iban teniendo y las cualidades sistémicas requeridas se fue armando el documento de Descripción de la Arquitectura. Un breve resumen del mismo se puede observar en el punto anterior donde se visualizan los componentes más importantes de la solución.

Podemos mencionar que a diferencia de otras metodologías ágiles (XP entre las más conocidas) que plantean el no realizar ningún esfuerzo de arquitectura al principio, AgEnD sugiere especificar las cuestiones técnicas más importantes en forma temprana de manera de minimizar el impacto del cambio en este sentido. Esto permitió que una vez elegida la arquitectura, el Arquitecto y los desarrolladores construyeron un prototipo proof of concept que validaría las decisiones tomadas. Se seleccionaron los casos de uso más críticos de los que estaban especificados y se implementó el prototipo. El mismo fue exitoso y sentó la base para la posterior implementación de los casos de uso.

Integración Continua

De acuerdo a la práctica de AgEnD se configuró un entorno de desarrollo que fomentara la integración continua. Los Desarrolladores trabajaban en la implementación de los casos de uso y periódicamente subían su trabajo al repositorio, en este caso el CVS. El código sólo era subido si las pruebas unitarias pasaban en su totalidad.

Una vez que estaba subida la aplicación la misma era testeada unitariamente y funcionalmente por el desarrollador para verificar que el sistema continuaba en funcionamiento. Al fin de cada semana el Tester tomaba todo lo construido y guiándose con los casos de uso llevaba a cabo el testing funcional en forma conjunta con el Cliente en algunos casos.

Gracias a la práctica de Integración Continua se evitaron los problemas de modelos anteriores más rigurosos que posponían la integración para el final del proyecto, una vez que todos los módulos estuvieran codificados, lo cual creaba un alto riesgo debido a la incompatibilidad de interfases y comportamientos. En el proyecto de FIDoNET no existieron problemas de este tipo.

Peopleware

El énfasis de AgEnD en la gente fue llevado a la práctica en el proyecto. En todo momento el Líder de Proyecto fomentaba la motivación de su equipo. Hubo solamente un par de semanas con algunos días con horas extras, especialmente en fechas críticas de entregas pero los mismo fueron seguidos de semanas más tranquilas con menos carga laboral.

Las personas del equipo disponían de amplios lugares de trabajo, máquinas potentes, cómodos escritorios que les permitían llevar a cabo programación de a pares si se deseaba. También permitía que el Cliente pudiera sentarse con el Analista a evaluar la aplicación.

En todo momento se fomentó la cohesión del grupo llevando a cabo actividades extra-curriculares que permitieran generar un espíritu de grupo. Estas actividades incluyeron salidas después del trabajo, asados con el equipo de desarrollo, almuerzos a cargo de la empresa, etc. Las mismas ayudaron a que la gente entrara en contacto entre sí y se hablarán de cuestiones no concernientes al ámbito laboral.

Administración del Conocimiento

Como se mencionó anteriormente el propósito de una metodología es garantizar el tener una forma de trabajo predecible que permita terminar en costo, tiempo y forma los proyectos. Dado que para la siguiente versión de la aplicación era probable la

incorporación de más gente de la empresa resultaba de gran importancia llevar a cabo la práctica de Administración del Conocimiento.

El Líder de Proyecto era el encargado de administrar el conocimiento almacenado y guardarlo en un lugar para su posterior recuperación y utilización. El propósito era administrar el conocimiento organizacional para que FIDoNET pudiese desarrollar aplicaciones más ambiciosas ya que la mayor parte del conocimiento estaba almacenado. La herramienta utilizada para el proyecto fue una Swiki – herramienta web que permite automatizar la generación, publicación y administración de contenido.

En cada una de las reuniones semanales de revisión de la iteración el Líder de Proyecto se encargaba de averiguar que conocimiento se había generado de cada participante y posteriormente se juntaba con dicha persona/s para capturarlo de una manera estandarizada.

Artefactos

De acuerdo a la definición de AgEnD se recomienda especificar al principio del proyecto aquellos artefactos que serán generados y mantenidos durante el ciclo de vida. Esta es una de las tareas que el Coordinador de Proceso (tomado en este caso por el Líder de Proyecto) realiza durante las primeras iteraciones. El resultado de la misma es el siguiente:

- *Minutas de Reunión*, documentos que ponen de manifiesto los tópicos que fueron tratados en una reunión de FIDoNET
- *Plan de Proyecto*, representado mediante un diagrama de Gantt en el que se irá plasmando la planificación del proyecto durante el tiempo
- *Visión*, contiene los features principales del sistema así como la identificación de los involucrados en el proyecto
- *Guía de Estilo*, contiene las decisiones que fueron tomadas en el diseño de la UI y los estándares que se deberán tener en cuenta para la extensión de este diseño
- *Casos de Uso*, contiene los requerimientos funcionales del sistema con apéndices de diseño

- *Documento de Arquitectura*, contiene las decisiones de diseño más relevantes que se tomaron en el proyecto y las vistas arquitectónicas relevantes
- *Código Fuente y Scripts de Despliegue*, incluye el código de las clases, las páginas HTML, las imágenes, los archivos de configuración, los scripts de despliegue y demás recursos que componen el proyecto de desarrollo
- *Repositorio Unificado*, repositorio unificado implementado en CVS que contendrá la totalidad de los ítems de configuración del proyecto (documentos, código, imágenes, videos, etc.)
- *Planilla de Incidentes*, usada por el Tester al momento de reportar bugs y mejoras a los Desarrolladores

Conclusiones acerca del Proyecto

Después de seis meses de desarrollo el proyecto llega a un hito importante, la entrega del release 1.0 que concluye la puesta en práctica de AgEnD. El mismo ha terminado en forma satisfactoria para las partes involucradas. Se han implementado todas las prácticas recomendadas por AgEnD y se realizó una reunión de Cierre de esta primer etapa para relevar y analizar el proyecto y la metodología.

Experimentación de Prácticas utilizando AgEnD

Segundo Ejemplo: Aplicación en CONEST

Tomando otro proyecto de análisis mostraremos cómo las prácticas recomendadas por AgEnD permitieron llegar al éxito. Una diferencia importante es que no se implementaron tantas prácticas como en el proyecto descrito anteriormente.

Este proyecto fue encarado por una empresa que presta servicios de desarrollo de software. El cliente para el cual realiza el proyecto en particular es CONEST[10] quien maneja importantes centros de compras dentro del país. CONEST ha decidido encargarle a la consultora un sistema para la administración de recursos de hardware y software disponibles en la empresa. Dicho sistema será internamente por el departamento de tecnología de CONEST quién deberá administrar el inventario de estos bienes. El proyecto es relativamente pequeño (3 meses aprox.) como se mostrará en la sección de estimaciones.

Background del Proyecto

CONEST ha decidido utilizar un equipo de desarrollo que inicialmente será de 3 personas, y posteriormente en la construcción crecerá a unas 5 personas. El proyecto será conducido utilizando algunas prácticas de AgEnD como proceso, siendo las prácticas seleccionadas por un Coordinador con conocimiento de la metodología (el Autor en este caso). Al igual que antes recalcamos la realización de reuniones frecuentes para configurar AgEnD acorde a las necesidades del equipo de desarrollo. Todo el equipo detrás del emprendimiento de CONEST está comprometido a utilizar AgEnD en todas sus fases pudiendo adaptarlo a medida se avanza en el desarrollo mediante la intervención del Coordinador.

Roles y Recursos

Los roles de AgEnD están llevados a cabo por distintas personas del proyecto. A continuación mencionamos a los trabajadores:

[10] Nuevamente, por cuestiones de confidencialidad se dará un nombre de fantasía de la empresa.

- Existen un *Usuario Experto*, el cual trabaja en las oficinas del Cliente y forma parte del departamento de tecnología este interactúa únicamente con el *Analista Funcional*. A lo largo de este capítulo nos referimos a este rol como “el Cliente”.

  Reseña: No existe mucha referencia del Usuario Experto ya que es el primer proyecto en que este contrata a la consultora. Sin embargo de los primeros relevamientos surge que son personas flexibles que ayudarán al equipo en todas las cuestiones funcionales.

- Existe un *Ejecutivo de Cuenta* que velará por mantener el proyecto en su rumbo, eliminando obstáculos políticos que estén fuera del alcance del *Equipo de Desarrollo.*

  Reseña: El *Ejecutivo de Cuenta* es la persona que efectúa el primer contacto con el Cliente. Sus aptitudes son las de ser un muy buen negociador y conocer el área desde un punto de vista comercial.

- Existe un *Líder de Proyecto.*

  Reseña: El Líder de Proyecto es la persona encargada de gestionar el proyecto. El mismo tiene otros proyectos a su cargo con lo cual su dedicación será part-time.

- Existe una *Analista Funcional* encargada de llevar a cabo el relevamiento.

  Reseña: La Analista es una persona con mucha habilidad y experiencia en relevamientos en grandes proyectos. Esta especializada en la confección de Casos de Uso y en el uso de herramientas CASE como el Enterprise Architect para el modelado. La misma cumple sus funciones en otros proyectos con lo cual su dedicación será part-time.

- Existe un *Coordinador* que ayudará al equipo en la implementación de la metodología.

  Reseña: El Coordinador ayudará al Equipo de Desarrollo en la implementación de las prácticas de AgEnD y en la creación de los artefactos propuestos. También actuará de Mentor en relación a las

tecnologías seleccionadas en el proyecto. Esta persona cumple sus funciones en otros proyectos con lo cual su dedicación será part-time.

- Existen dos *Desarrolladores* que poseen distintos grados de experiencia en las tecnologías del proyecto. Uno estará involucrado en el proyecto desde el principio y el otro lo iniciará posteriormente.

  Reseña: Los dos tienen cargos de Programadores Junior y conocen a en forma teórica la tecnología seleccionada contando con poca experiencia práctica. No poseen abundante experiencia laboral.

- Existe un *Tester* que se encargará de diseñar y ejecutar los casos de prueba de la aplicación

  Reseña: El Tester será el responsable de llevar a cabo el control de calidad de la aplicación. A medida se vayan liberando versiones realizará el testing funcional de las mismas en base a los casos de prueba que este construyo.

Tecnologías

Al momento de armar la Propuesta Económica del proyecto se priorizó la selección de tecnologías basadas sobre la plataforma Java 2, prefiriéndose aquellas que respondiesen al modelo de desarrollo Open Source y que pudiesen usarse libremente sin costos de licencias asociados.

Arquitectura de la Aplicación

Arquitectura Web: arquitectura que presenta la interfase al usuario mediante un navegador de internet pudiendo desacoplar las capas de interfase/lógica de negocio/datos en el servidor mediante frameworks disponibles.

La arquitectura de la aplicación para prestar servicios web estará compuesta por las siguientes capas (tiers):

• Capa de Presentación: páginas web servidas mediante un framework de MVC que desacopla esta capa de las capas de lógica del negocio y manejo de datos.

• Capa de Lógica del Negocio: inicialmente, se puede pensar en un esquema sencillo de lógica del negocio que simplemente reciba/envíe la información a la capa de presentación, mediante la interacción directa con la capa de manejo de datos. Se planteará un esquema de componentes que resuelvan los pedidos realizados sin necesidad de utilizar un application server.

• Capa de Manejo de Datos: para esto se utilizará un motor de base de datos relacional. El mismo será accedido mediante el patrón DAO (Data Access Object) que desacopla la persistencia del RDBMS seleccionado.

Plataforma Base y Herramientas

Microsoft Windows: SO de desarrollo

Apache Tomcat: Web Container

CVS: Sistema de Administración de la Configuración

Eclipse: IDE de Desarrollo

Macromedia Dreamweaver: Herramienta de diseño de páginas web

Enterprise Architect: Herramienta de modelado para realizar el Modelo de casos de uso y los diagramas de arquitectura

Motores de BD

Microsoft SQL Server: uno de los motores comerciales más utilizados en el mercado. El mismo es estándar del cliente por lo que resulta en una restricción de la aplicación. También se debe acceder al mismo únicamente mediante Stored Procedures que serán escritos por los desarrolladores.

Framework de MVC

Jakarta Struts: framework utilizado para la capa de presentación de la aplicación. El mismo se utiliza para armar el patrón MVC pudiendo desacoplar la presentación de la lógica del negocio.

Framework de Reporting (Descartado)

Jasper: inicialmente se decidió utilizar este framework de reporting que permite generar reportes customizados en la aplicación. El mismo está desarrollado en Java y es Open Source. Finalmente y por cuestiones de falta de features y problemas técnicos, la utilización del mismo fue descartada.

Data Access Object (DAO)

DAO: patrón de diseño a ser implementado en la solución. El mismo especifica una framework que se utiliza para realizar la capa de persistencia de la aplicación. El mismo provee una forma de acceder al motor de RDBMS sin necesidad de escribir código SQL dentro de la aplicación.

Como se observa en la Figura 038, tomada del libro sobre patrones en J2EE [Alur, 2001], el patrón encapsula la creación de entidades que mantienen los datos de la aplicación. Estos *Value Objects* son los que contienen las sentencias de accesos al RDBMS.

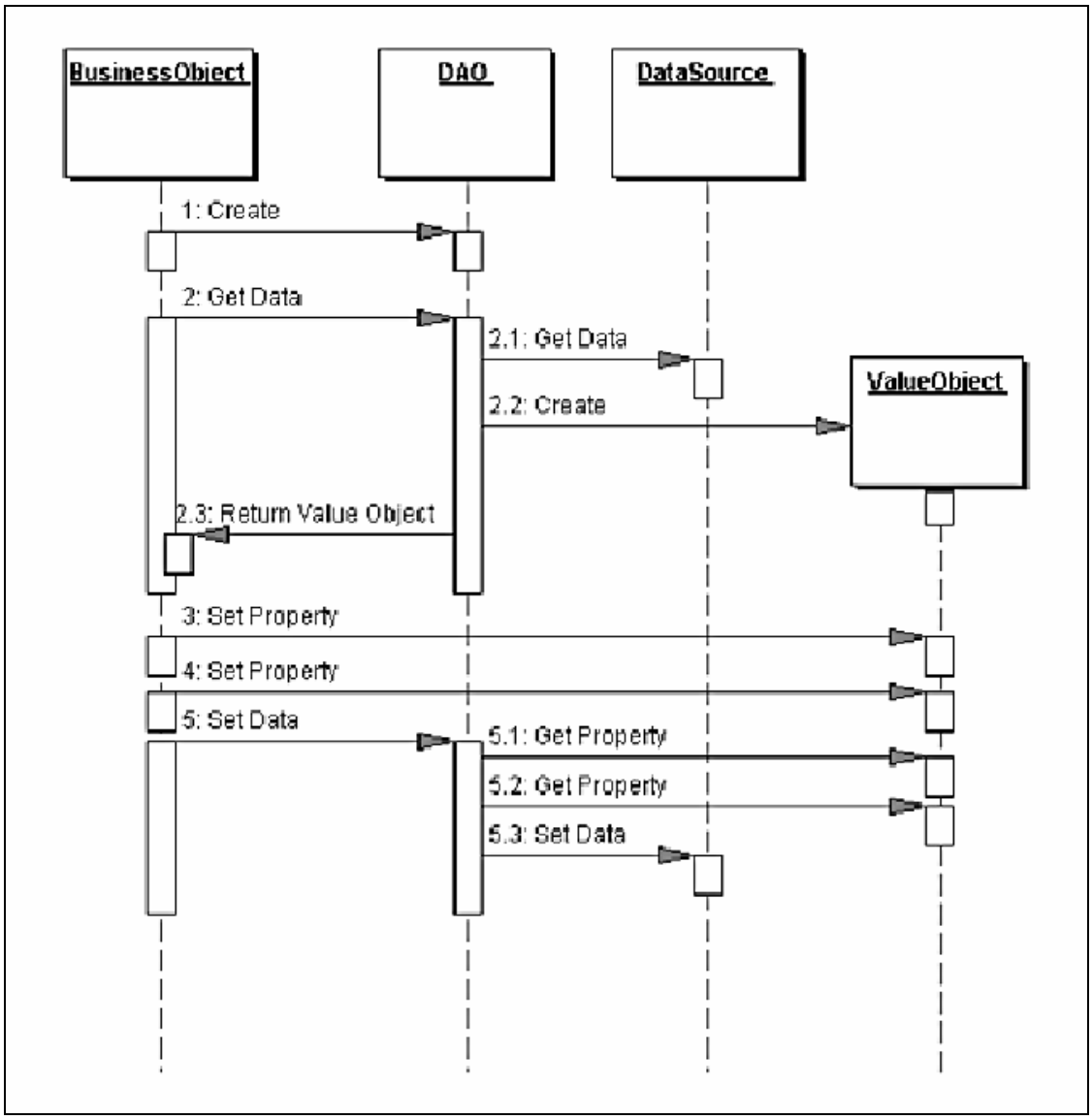

Figura 038. Diagrama de secuencia de interacciones en el patrón DAO. Tomada de [Alur, 2001].

Middlegen – DAO Generator: se utilizó una aplicación escrita en Java denominada Middlegen que automatiza la creación de todas las clases del patrón DAO tomando como input las tablas de la base de datos. De esta forma se redujeron considerablemente los tiempos de desarrollo de la capa de persistencia.

Lanzamiento del Proyecto

CONEST inicia el proyecto convocando a la consultora de sistemas para el desarrollo de su sistema. La consultora efectúa un primer relevamiento en el que obtiene los casos de uso de la aplicación. A partir de ahí elabora una propuesta económica con

una estimación tentativa, costos, equipo de desarrollo y demás información para el Cliente. El relevamiento es conducido por la Analista Funcional la cual identifica de forma temprana a los stakeholders del futuro sistema y los servicios que estos requerirán del sistema.

En la Propuesta entregada al Cliente se establece una planificación con las fases en que el proyecto de CONEST será dividido tomando como referencia las fases de AgEnD. La estimación se lleva a cabo mediante el conteo por UCP (Use Case Points) analizando la complejidad de los casos de uso en función de los escenarios planteados.

Resultados del Proyecto

Después de 3 meses y 3 semanas de desarrollo el proyecto llega al hito de entrega del release 1.0 que concluye la puesta en práctica de las prácticas de AgEnD. Si bien se ha sufrido un leve retraso debido a causas que mencionaremos a continuación, el proyecto entregó la funcionalidad requerida por el Cliente con la calidad acordada. Se han implementado gran parte de las prácticas recomendadas por AgEnD y se realizó una reunión de Cierre de esta primer etapa para relevar y analizar el proyecto y la metodología. A continuación se comentan aspectos de la metodología que contribuyeron al éxito del proyecto.

Configuración del Proceso

En el proyecto de CONEST una de las primeras cuestiones que se observó fue la carencia de una persona que actuara como Arquitecto en el proyecto tomando los requerimientos no funcionales y seleccionando una solución adecuada al problema planteado. Debido a esto, el Coordinador en sus tiempos libres ayudó a uno de los desarrolladores a armar dicho documento. Dado que éste no disponía de mucho tiempo se tomó algunos días para elaborar un Documento de arquitectura que comunicara la solución a los desarrolladores de manera fácil y sin necesidad de tener la presencia física del Arquitecto constantemente.

Al principio, esto sirvió al propósito, pero con el correr de las semanas se observó en las reuniones de avance diarias que los desarrolladores trabajaban demasiado lento debido a que la arquitecta les era nueva. Finalmente, siguiendo la práctica de

Máxima Comunicación el Coordinador dedicó unos 4 días full time al proyecto para llevar a cabo un refactoring de la Arquitectura suplantando el documento en sí por código ejecutable que sirviera para guía a los desarrolladores. Los resultados fueron excelentes.

Al igual que en el proyecto anterior se decidió agregar dentro de la disciplina de Administración de Proyecto del proceso al *Scrum Daily Meeting*. La misma era coordinada por el Líder de Proyecto quien preguntaba a cada persona del equipo qué había hecho la durante el día anterior, que iba a hacer durante ese día y que obstáculos tenía para llevar a cabo sus actividades. Esto permitió ir teniendo un control y monitoreo constante sobre el progreso y los riesgos que se iban dando.

En relación a los artefactos sugeridos por AgEnD la Visión fue suplantada por la Propuesta Económica a partir de la cual el Cliente decidía respecto a la ejecución del proyecto. Todos los demás artefactos sugeridos fueron realizados incluyéndose como parte de los casos de uso los prototipos de pantalla en HTML correspondientes. Esto sirvió para obtener un rápido feedback del usuario respecto a aquellas funcionalidades que resultaban ambiguas.

Administración de la Configuración

Se armó el repositorio en CVS que es la herramienta de SCM usada por la consultora. Todos los artefactos generados durante el proyecto fueron puestos bajo control de versiones. Para comodidad del equipo de desarrollo se decidió tener dos módulos independientes dentro de un mismo repositorio. El repositorio tenía el nombre del proyecto y existían dos módulos: uno denominado *Project* y otro *Sources*. El primer módulo incluía toda la documentación asociada a las distintas disciplinas de AgEnD categorizadas en directorios con el nombre de la disciplina. El segundo módulo tenía todo el código de la aplicación a ser cargado en la IDE utilizada – el Eclipse.

El mantener todo el código fuente bajo control de versiones permitió que en un punto los desarrolladores entregaran una primera versión de demo al cliente para validar los flujos de los casos de uso, usando el HEAD del repositorio para mantener los ítems de configuración. En paralelo, el Coordinador que emprendió tareas de Arquitectura empezó a trabajar en un refactoring de la Arquitectura destinado a mejorar la flexibilidad y mantenibilidad del código generado. Para esto se decidió abrir un branch

de desarrollo paralelo que permitiera desarrollar manteniendo limpio el HEAD para correcciones sobre las versiones de entrega al Cliente. Finalmente, dado que el refactoring de arquitectura resultó exitoso el desarrollo prosiguió por el branch quedando el HEAD obsoleto.

Máxima Comunicación

La práctica de Máxima Comunicación no pudo ser implementada completamente debido a que el Cliente estaba separado en su propia empresa y sólo podía ser accedido vía email o telefónicamente. Esto traía aparejado una lentitud de respuesta y una falla comunicacional importante que trató de ser subsanada mediante casos de uso validados y detallados por el Analista junto al Cliente. Ocurrieron situaciones en las que se tuvo que esperar casi 72 horas hasta obtener clarificaciones respecto a cuestiones de la aplicación por parte del Cliente.

Todas las personas del equipo de desarrollo estaban en un mismo piso de un edificio en oficinas adyacentes. Esto permitía que cualquier duda planteada por cualquier parte se resolviera mediante la rápida consulta con el miembro que tenía el conocimiento.

Enfoque en la Arquitectura

Esta práctica fue de gran ayuda en el desarrollo ya que permitió definir cuestiones técnicas que impactarían positivamente en la solución al problema. Sin embargo como ya fue mencionado fue donde se materializaron importantes riesgos que llevaron al desvío en el cronograma del proyecto.

Una vez que se comenzó con la especificación de los primeros casos de uso, en paralelo el Coordinador en rol de Arquitecto junto con los Desarrolladores comenzaron a definir la arquitectura del sistema. Uno de los requerimientos no funcionales más importante era el construir un sistema mantenible y flexible, el cual pudiera incorporar cambios con facilidad y que debería ser interoperable con otras aplicaciones dentro de los sistemas del Cliente.

Tomando como input los casos de uso que se iban teniendo y las cualidades sistémicas requeridas se fue armando el documento de Descripción de la Arquitectura.

Un breve resumen del mismo se puede observar en el punto anterior donde se visualizan los componentes más importantes de la solución.

El único comentario que podemos incluir es que la falta de una persona experimentada desde el punto de vista técnico guiando al equipo de desarrollo full time, como establece el rol de Arquitecto en AgEnD, causó problemas debido a la inexperiencia del equipo de desarrollo. Volvemos a destacar la necesidad de invertir constantemente en la gente y el riesgo que trae aparejado el no tener recursos a la altura de las circunstancias.

Estimaciones Ágiles

Esta práctica fue utilizada al principio del proyecto en el armado de la Propuesta. Durante la primera fase se tomaron los casos de uso del producto y el Líder de Proyecto en forma conjunta con el Coordinador y el Cliente, dieron prioridad a seleccionando la versión del producto en que cada uno aparecería. Siguiendo a AgEnD se tomaron los casos de uso seleccionados para la primera versión de manera de efectuar una estimación global de la duración del proyecto. Se dejó en claro que dicha estimación arrojaría un resultado aproximado y dada la naturaleza de los procesos ágiles el mismo simplemente serviría para planificar hitos importantes del proyecto.

Usando la metodología de UCP (Use Case Points) junto con los factores de ajuste correspondientes y efectuando una estimación en paralelo basada en la experiencia de especialistas técnicos dentro de la consultora la estimación arrojó un esfuerzo para el proyecto de 12 meses/persona. De acuerdo al equipo de desarrollo descrito anteriormente se estimó la duración en 3 meses.

Este resultado fue bastante acertado si se tiene en cuenta que la primer versión fue liberada 3 meses y 3 semanas después. Cabe remarcar el desvío surgido debido a los temas antes mencionados respecto a la comunicación de la arquitectura y a las dificultades con el paquete de reporting seleccionado.

Integración Continua

De acuerdo a la práctica de AgEnD se configuró un entorno de desarrollo que fomentara la integración continua. Los Desarrolladores trabajaban en la implementación

de los casos de uso y periódicamente subían su trabajo al repositorio, en este caso el CVS.

Una vez que estaba subida la aplicación la misma era testeada unitariamente y funcionalmente por el desarrollador para verificar que el sistema continuaba en funcionamiento. Al fin de cada iteración el Tester tomaba todo lo construido y guiándose con los casos de prueba llevaba a cabo el testing funcional en forma conjunta con el Cliente en algunos casos. En el proyecto de CONEST no existieron problemas relacionados con la integración.

Peopleware

El énfasis de AgEnD en la gente fue llevado a la práctica parcialmente en el proyecto. En todo momento el Líder de Proyecto fomentaba la motivación de su equipo. Hubo solamente un par de semanas con algunos días con horas extras, especialmente en fechas críticas de entregas pero los mismo fueron seguidos de semanas más tranquilas con menos carga laboral.

Las personas del equipo disponían de amplios lugares de trabajo, máquinas potentes, cómodos escritorios que les permitían llevar a cabo programación de a pares si se deseaba.

Los mayores inconvenientes en relación a esta práctica fueron la carencia de una persona cubriendo el rol de Arquitecto en forma continua y la falta de un alto nivel de comunicación con el Cliente. Respecto a esto último toda la comunicación con el Cliente era realizada mediante la intermediación del Analista Funcional con lo cual el delay resultaba muy ineficiente para el desarrollo. Esto fue notificado por el Coordinador en forma temprana ya que implicaba un alto riesgo en la implementación de una metodología ágil pero sin embargo el Líder de Proyecto no pudo efectuar ninguna acción correctiva más que tratar de mantener un canal abierto con un usuario experto para posibles consultas.

Administración del Conocimiento

El Coordinador era el encargado de administrar el conocimiento almacenado y guardarlo en un lugar para su posterior recuperación y utilización. El propósito era

administrar el conocimiento organizacional para que CONEST pudiese desarrollar aplicaciones más ambiciosas ya que la mayor parte del conocimiento estaba almacenado. Dado que la arquitectura era bastante conocida y se contaba con suficiente información se decidió simplemente documentar el funcionamiento de aquellos paquetes sobre los cuales no se disponía de información.

El Coordinador pidió a los Desarrolladores que armarán un documento como introducción al Jasper – el framework de reporting que fue analizado pero descartado finalmente, y otro documento que sirviera de introducción para el uso del Middlegen – el paquete que permitía automatizar la creación de DAOs a partir de una base de datos determinada.

Una vez que dichos documentos fueron generados el Coordinador los formateó adecuadamente y los puso a disposición de todos en la intranet de conocimiento de la consultora. Esta medida llevada a la escala de todos los proyectos en que se implemente AgEnD permitirá que cualquier tipo de conocimiento generado sea permeado en la gente aumentando el conocimiento organizacional.

Artefactos

De acuerdo a la definición de AgEnD se recomienda especificar al principio del proyecto aquellos artefactos que serán generados y mantenidos durante el ciclo de vida. Esta es una de las tareas que el Coordinador de Proceso realiza durante las primeras iteraciones. El resultado de la misma es el siguiente:

- *Minutas de Reunión*, documentos que ponen de manifiesto los tópicos que fueron tratados en una reunión de CONEST
- *Plan de Proyecto*, representado mediante un diagrama de Gantt en el que se irá plasmando la planificación del proyecto durante el tiempo
- *Propuesta Económica*, contiene los features principales del sistema así como la identificación de los involucrados en el proyecto, el costo total, el equipo de desarrollo y una estimación inicial del mismo
- *Prototipos de GUI*, realizados por un diseñador gráfico para todas las pantallas de cada transacción dentro de un caso de uso

- *Casos de Uso*, contiene la especificación de los requerimientos funcionales del sistema
- *Documento de SRS*, contiene los requerimientos funcionales de carácter más técnico y no funcionales del sistema con apéndices de diseño
- *Documento de Arquitectura*, contiene las decisiones de diseño más relevantes que se tomaron en el proyecto y las vistas arquitectónicas relevantes
- *Código Fuente y Scripts de Despliegue*, incluye el código de las clases, las páginas HTML, las imágenes, los archivos de configuración, los scripts de despliegue y demás recursos que componen el proyecto de desarrollo
- *Repositorio Unificado*, repositorio unificado implementado en CVS que contendrá la totalidad de los ítems de configuración del proyecto (documentos, código, imágenes, videos, etc.)
- *Casos de Prueba*, contienen los flujos de ejecución que serán utilizados por los Testers para probar la aplicación desde un punto de vista funcional; son creados a partir de los casos de uso
- *Planilla de Incidentes*, usada por el Tester al momento de reportar bugs y mejoras a los Desarrolladores

Conclusiones acerca del Proyecto

Después del tiempo mencionado de desarrollo el proyecto llega a un hito importante, la entrega del release 1.0 que concluye la puesta en práctica de AgEnD. El mismo ha terminado en forma satisfactoria para las partes involucradas, a excepción del desvío mencionado. Se han implementado muchas de las prácticas recomendadas por AgEnD y se realizó una reunión de Cierre de esta primer etapa para relevar y analizar el proyecto y la metodología.

Cabe mencionar que en aquellos casos en los que las prácticas no pudieron ser implementadas siguiendo los lineamientos propuestos en esta tesis, hubo importantes perjuicios que se manifestaron como desvíos en el cronograma del proyecto. Es de

opinión del autor que estas cuestiones no hubieran sucedido si las circunstancias hubieran permitido la implementación completa de las prácticas.

Capitulo V - Conclusiones

Las metodologías de desarrollo de software ágiles permiten a los pequeños grupos de desarrollo concentrarse en la tarea de construir software fomentando prácticas de fácil adopción y un entorno ordenado que ayude a que las personas trabajen mejor y permita que los proyectos finalicen exitosamente. Las mismas están basadas en los cuatro principios del Manifiesto Ágil que fueron mencionados al principio de este trabajo. AgEnD, la metodología propuesta en esta tesis, avanza en el conocimiento teórico de estos procesos analizando estos principios mencionados y reuniendo prácticas y patrones que contribuyen a la implementación y posterior adaptación del proceso a la realidad de cada organización.

Para llevar a cabo este trabajo se recabó información exhaustiva de procesos de desarrollo, áreas de conocimiento de ingeniería de software, metodologías ágiles, prácticas de desarrollo ágil y otras disciplinas dentro de la informática. Además, se leyó bibliografía de temas como Administración de Empresas, Psicología Sistémica y Recursos Humanos. Como se observará en la parte final de bibliografía, la lista que fue reunida no es menor y representa en gran parte el “state-of-the-art” de los conocimientos aplicados en esta tesis. Mencionaremos a continuación las tareas llevadas a cabo en el transcurso de este trabajo.

Primeramente, se describió la historia de los procesos de desarrollo de software, explicando brevemente los distintos modelos que fueron surgiendo y las virtudes / falencias de cada uno. A esto prosiguió una breve historia del surgimiento de las metodologías ágiles y una caracterización de las más importantes durante el período de escritura de esta tesis: XP, Scrum, Cristal Clear, DSDM, FDD, ASD, XBreed. Gracias a este capítulo logramos ubicar al lector dentro del universo del cual habríamos de explayarnos más adelante.

Posteriormente, en el capitulo 2, se describieron las problemáticas planteadas por los modelos de procesos de desarrollo existentes y cómo las metodologías ágiles, en particular AgEnD, proponían principios para mitigar los riesgos asociados a la complejidad inherente del desarrollo de software (descrita por Fred Brooks). Esta descripción pretende auxiliar al lector a entender el porqué de los principios sobre los

cuales se basan las metodologías ágiles y que dirigirán todos los objetivos de la solución propuesta más adelante.

En el capítulo 3 se comenzó a desglosar la metodología propuesta para su entendimiento en detalle. Partiendo de una definición de aspectos abarcativos de un proceso mencionada por Alistair Cockburn, el texto analiza los diversos elementos que componen AgEnD: roles, artefactos, fases, disciplinas, hitos, patrones de desarrollo. Manteniendo la ideología de priorizar a las personas sobre el proceso, se trato de poner continuamente el énfasis en el factor humano tratando de mantener el overhead metodológico al mínimo posible. Asimismo, todos los problemas planteados en el capitulo anterior tienen una respuesta tangible en la metodología. Se trató de aprender de los viejos errores y de generar un repositorio de buenas prácticas.

Avanzando en el capítulo se propusieron una serie de técnicas y estudios que permiten implementar una metodología ágil dentro de una organización. Esta parte constituye en sí uno de los principales aportes al espectro de metodologías ágiles por no encontrarse muchas líneas de investigación hasta el momento y menos que tomen en cuenta otras ramas del conocimiento distintas a la informática. Es creencia del autor que es de suma importancia: empezar a formalizar la mejora del proceso dentro de una organización a consecuencia del incremento continuo en la complejidad de los sistemas construidos y disponer de personas que se encarguen de velar por mantener y actualizar un proceso lo más ágil e institucionalizado posible – valor estratégico a futuro de los SEPG (Software Engineering Process Group) que ya se encuentran en las grandes organizaciones.

En el capítulo 4 se presentaron dos casos de éxito de implementación de AgEnD en distintas organizaciones. Estos proyectos contaron con la presencia constante del autor de la tesis, quien realizó la capacitación inicial de las personas respecto a AgEnD y quien continuamente recabó información de cómo el proceso era llevado a la práctica, adaptándolo de ser necesario. Estas puestas en práctica resultaron positivas ya que dieron una cierta tangibilidad a una tesis de ingeniería de software fuertemente orientada hacia aspectos conceptuales.

En relación con puntos de investigación a futuro el autor evaluará la posibilidad de crear una herramienta CASE que ayude a customizar un proceso ágil para un proyecto en particular. Asimismo, se iniciaron líneas de investigación en otras

disciplinas para analizar factores humanos dentro de la Ingeniería de Software. El profundizar más por la bibliografía y las áreas de conocimiento de la Sociología, Psicología, Relaciones Laborales ayudará a hacer crecer nuestra ingeniería.

Anexo A - Templates de Artefactos

A continuación se proponen distintos templates para los artefactos recomendados por AgEnD. Los mismos podrán ser tomados por el Coordinador como punto de partida al momento de implementar la metodología en la organización.

Los templates que se ofrecen son para los siguientes artefactos:

- Visión
- Lista de Riesgos
- Especificación de Casos de Uso
- Documento de Especificación de Requerimientos de Software (SRS)
- Descripción de la Arquitectura
- Casos de Prueba
- Nota de Entrega

Dado que los mismos son documentos totalmente independientes de esta tesis, se incluyen en los distintos formatos generados por cada herramienta al fin de la misma.

Anexo B - Tabla de Lenguajes de Programación

Release 8.2, Marzo 1996

Por Capers Jones, Chairman, Software Productivity Research, Inc.

Table 1. Language Level Relationship to Productivity

LANGUAGE LEVEL	PRODUCTIVITY AVERAGE PER STAFF MONTH
1 - 3	5 to 10 Function Points
4 - 8	10 to 20 Function Points
9 - 15	16 to 23 Function Points
16 - 23	15 to 30 Function Points
24 - 55	30 to 50 Function Points
Above 55	40 to 100 Function Points

What Is The Basis For Language Levels?

The languages and levels in Table 2 were gathered in four ways.

- Counting Function Points and Source Code
- Counting Source Code
- Inspecting Source Code
- Researching Languages

Counting Function Points And Source Code

Actual counts of Function Points and source code statements were performed. Samples of counting Function Points and source code statements were done on Ada, several BASIC

dialects, COBOL, PASCAL, and PL/I.

Counting Source Code

Source code statements were counted, then compared to the size of the same program in languages of known levels. Assembly, APL, C, OBJECTIVE C, FORTH, FORTRAN, LISP, PILOT, and PROLOG are languages that produce the same source code count as COBOL. So code sizes were compared to the known quantity of COBOL source code.

Inspecting Source Code

Source code inspection for common applications was done. Then the volume of code for the application in a measured language was hypothesized. ACTOR, CLARION, and TRUE BASIC are examples of languages that were inspected and their levels hypothesized by subjective means.

List Of Programming Languages

As of 1996, there were more than 500 languages and major dialects of languages available to software practitioners. Table 2 lists the most common of them in what is considered version 7 of the SPR Programming Languages Table.

Table 2. Programming Languages and Levels

LANGUAGE	*LEVEL*	*AVERAGE SOURCE STATEMENTS PER FUNCTION POINT*
1032/AF	20.00	16
1st Generation default	1.00	320
2nd Generation default	3.00	107
3rd Generation default	4.00	80
4th Generation default	16.00	20
5th Generation default	70.00	5
Access	8.50	38
Ada 83	4.50	71
Ada 95	6.50	49
AI shell default	6.50	49
ANSI BASIC	5.00	64
ANSI COBOL 74	3.00	107
ANSI COBOL 85	3.50	91
ANSI SQL	25.00	13

Application Builder	16.00	20
Application Manager	9.00	36
Assembly (Basic)	1.00	320
Assembly (Macro)	1.50	213
Associative default	5.00	64
awk	15.00	21
BASIC	3.00	107
BASIC A	2.50	128
Basic assembly	1.00	320
Berkeley PASCAL	3.50	91
BETTER BASIC	3.50	91
C	2.50	128
C Set 2	3.50	91
C++	6.00	53
DELPHI	11.00	29
Eclipse	6.50	49
EIFFEL	15.00	21
EXCEL 1-2	51.00	6
EXCEL 3-4	55.00	6
EXCEL 5	57.00	6
FOXPRO 2.5	9.50	34
GENEXUS	21.00	15
Haskell	8.50	38
HTML 2.0	20.00	16
HTML 3.0	22.00	15
IBM CICS/VS	8.00	40
IBM Compiled BASIC	3.50	91
IBM VS COBOL	3.00	107
IBM VS COBOL II	3.50	91
INFORMIX	8.00	40
JAVA	6.00	53
JCL	1.45	221
LISP	5.00	64
LOTUS 123 DOS	50.00	6
LOTUS Macros	3.00	107
Machine language	0.50	640
Macro assembly	1.50	213
MATHCAD	60.00	5
Microsoft C	2.50	128
MODULA 2	4.00	80
MOSAIC	50.00	6
MS C ++ V. 7	6.00	53
MS Compiled BASIC	3.50	91

Natural language	0.10	3200
Non-procedural default	9.00	36
Notes VIP	9.00	36
Object-Oriented default	11.00	29
OBJECT Assembler	5.00	64
Object LISP	11.00	29
Object LOGO	11.00	29
Object PASCAL	11.00	29
ORACLE	8.00	40
Oracle Developer/2000	14.00	23
PARADOX/PAL	9.00	36
PASCAL	3.50	91
PERL	15.00	21
Persistance Object Builder	15.00	21
PILOT	6.00	53
PL/I	4.00	80
PL/M	4.50	71
PL/S	3.50	91
PowerBuilder	20.00	16
POWERHOUSE	23.00	14
PPL (Plus)	8.00	40
Pro-C	12.00	27
PRO-IV	5.50	58
Problem-oriented default	4.50	71
Procedural default	3.00	107
Professional PASCAL	3.50	91
Program Generator default	20.00	16
PROGRESS V4	9.00	36
PROLOG	5.00	64
QBasic	5.50	58
QUATTRO	51.00	6
QUATTRO PRO	51.00	6
Query default	25.00	13
QUICK BASIC 1	5.00	64
QUICK BASIC 2	5.25	61
QUICK BASIC 3	5.50	58
Quick C	2.50	128
Quickbuild	11.50	28
QUIZ	22.00	15
Reuse default	60.00	5
REXX (MVS)	4.00	80
REXX (OS/2)	7.00	46
RPG I	4.00	80

RPG II	5.50	58
RPG III	5.75	56
Screen painter default	57.00	6
SIMULA 67	7.00	46
Simulation default	7.00	46
SMALLTALK 286	15.00	21
SMALLTALK 80	15.00	21
SMALLTALK/V	15.00	21
SQL	25.00	13
SQL-Windows	27.00	12
Statistical default	10.00	32
Strongly typed default	3.50	91
Symantec C++	11.00	29
Tandem Access Language	3.50	91
Topspeed C ++	11.00	29
Turbo C	2.50	128
TURBO C++	6.00	53
TURBO EXPERT	6.50	49
Visual Basic 1	7.00	46
Visual Basic 2	7.50	43
Visual Basic 3	8.00	40
Visual Basic 4	9.00	36
Visual Basic 5	11.00	29
Visual Basic DOS	8.00	40
Visual C++	9.50	34
Visual COBOL	16.00	20
Visual Objects	20.00	16
VisualAge	15.00	21
VisualGen	18.00	18

Anexo C - Glosario

Casos de Uso. Notación utilizada para representar los requerimientos funcionales de un sistema basada en el esquema propuesto por Ivar Jacobson a principios de la década del ‘90.

CVS (Concurrent Versions System). Herramienta que permite seguir los cambios a un conjunto de archivos a lo largo del tiempo. CVS es comúnmente usado en el desarrollo de software para:

- permitir que múltiples desarrolladores coordinen sus cambios
- poder realizar el seguimiento de todas las versiones de los ítems de configuración
- fomentar el desarrollo en simultáneo de diferentes versiones del mismo código

IS/IT (Information Systems/Information Technology). Refiérase a los sectores de las organizaciones relacionados con el manejo de la información y con las tecnologías involucradas en este manejo.

Metáfora. Equivalente de la arquitectura en el universo de XP. La metáfora guía al desarrollo proveyendo de integridad conceptual al diseño y comunicando a todos los involucrados los elementos básicos de la solución y sus relaciones.

Product Backlog. En el universo de Scrum este es el conjunto de requerimientos a ser implementados para el sistema en construcción siendo el mismo priorizado continuamente por el Cliente para armar el Backlog de cada iteración (denominado Sprint Backlog).

Programación de a Pares (Pair Programming). Una de las doce practicas de XP, en la que dos personas programan frente a una computadora. Una de ellas escribe el código mientras la otra inspecciona continuamente lo que se desarrolla realizando un Control de Calidad en el momento.

Refactoring. Técnica que mantiene intacto el funcionamiento del software mejorando la estructura interna del mismo.

ROI (Return On Investment). Métrica financiera que representa el retorno en ganancias que se obtendrá posteriormente a realizar una inversión dada.

Sprint. En el universo de Scrum este es el nombre que se le da a una iteración.

Stakeholder. Toda aquella persona u organización siendo influenciada o ejerciendo influencia sobre el software que está siendo construido.

Unit Testing. Técnica utilizada por muchas de las metodologías ágiles descritas que consiste en realizar pruebas automatizadas que verifiquen el correcto funcionamiento de las clases que son desarrolladas. Para esto se utilizan frameworks como el JUnit, Cactus, HttpUnit, etc.

WBS. Sigla en inglés de un *Work Breakdown Structure*. Especifica una técnica que consiste en desglosar un proyecto en actividades atómicas y estimar el esfuerzo de cada una de estas actividades. Finalmente, mediante la sumatoria de todos los esfuerzos se tendrá el esfuerzo del proyecto.

Referencias Bibliográficas

[Alexander, 1977] Alexander, Christopher, *A Pattern Language*, New York, Oxford University Press, 1977.

[Alexander, 1979] Alexander, Christopher, *The Timeless Way of Building*, New York, Oxford University Press, 1979.

[Alur, 2001] Alur, Deepak, John Crupi, Dan Malks, *Core J2EE Patterns*, New York: Prentice Hall, 2001.

[Beck, 2000] Beck, Kent, *Extreme Programming Explained*, Addison-Wesley The XP Series, 2000.

[Beck, 2002] Beck, Kent, *Test-Driven Development By Example*, Addison-Wesley, November 2002.

[Boehm, 1981] Boehm, Barry W., *Software Engineering Economics*, Prentice Hall, 1981.

[Boehm, 1988] Boehm, Barry W., "A Spiral Model of Software Development and Enhancement," *IEEE Computer*, Vol. 21, No. 15 (May 1988) pp. 61-72.

[Boehm, 1995] Boehm, Barry W., *Anchoring the Software Process*, USC, November 1995.

[Boehm, 2000] Boehm, Barry W., Ellis Horowitz, Ray Madachy, Donald Reifer, Bradford K. Clark, Bert Steece, A. Winsor Brown, Sunita Chulani, Chris Abts, *Software Cost Estimation with Cocomo II*, Prentice Hall, 2000.

[Booch, 1999] Booch, Grady, Ivar Jacobson, James Rumbaugh, *The Unified Modeling Language User Guide*, Addison-Wesley Object Technology Series, 1999.

[Brooks, 1975] Brooks, Frederick P., *The Mythical Man Month*, Reading, Mass.: Addison-Wesley, 1975.

[Brooks, 1987] Brooks, Frederick P., "No Silver Bullet: Essence and Accidents of Software Engineering," *IEEE Computer*, Vol. 20, No. 4 (April 1987) pp. 10-19.

[C3, 1998] C3 Team, "Chrysler Goes to "Extremes"," *Distributed Computing*, (October 1998) pp. 24-28.

[Calligo, 2003] Calligo, Aníbal, Marcelo Schenone, *Técnicas de Producción de Software I – Modelos de Proceso de Desarrollo*, Facultad de Ingeniería, UBA, 2003.

[Charvat, 2003] Charvat, Jason, *Project Management Methodologies: Selecting, Implementing, and Supporting Methodologies and Processes for Projects*, John Wiley & Sons, 2003.

[Coad, 1998] Coad, Peter, Eric Lefebvre, Jeff De Luca, *Feature-Driven Development*, http://www.cs.jhu.edu/~scott/oos/software/togetherj/help/Users-Guide/Feature_Driven_Development.htm, 1998.

[Coad, 1999] Coad, Peter, Eric Lefebrve, Jeff De Luca, *Java Modeling in Color with UML*, Prentice Hall, 1999.

[Coad, 2000] Coad, Peter, *Feature-Driven Development and Extreme Programming*, http://www.togethercommunity.com/coad-letter/Coad-Letter-0070.html, Coad Letter, Issue 70, July 2000.

[Cockburn, 1998] Cockburn, Alistair, *Surviving Object Oriented Projects*, Addison-Wesley Object Technology Series, 1998.

[Cockburn, 1999] Cockburn, Alistair, *Characterizing People as Non-Linear, First-Order Components in Software Development*, http://members.aol.com/humansandt/papers/nonlinear/nonlinear.htm, October 1999.

[Cockburn, 2000a] Cockburn, Alistair, *Writing Effective Use Cases*, Addison-Wesley, 2000.

[Cockburn, 2000b] Cockburn, Alistair, *Just-In-Time Methodology Construction*, http://crystalmethodologies.org/articles/jmc/justintimemethodologyconstruction.html, January 2000.

[Cockburn, 2001a] Cockburn, Alistair, *Agile Software Development*, Addison-Wesley, The Agile Software Development Series, 2001.

[Cockburn, 2001b] Cockburn, Alistair, *Crystal "Clear": A human-powered software development methodology for small teams*, http://members.aol.com/humansandt/crystal/clear/, Humans and Technology, 1998-2001.

[Constantine, 1995] Constantine, Larry, *Constantine on Peopleware*, Yourdon Press Computing Series, Prentice Hall, 1995.

[Constantine, 2001] Constantine, Larry, *Process Agility and Software Usability: Toward Lightweight Usage-Centered Design*, Constantine & Lockwood, Ltd., University of Technology, Sydney, 2001.

[Crispin, 2002] Crispin, Lisa, Tip House, *Testing Extreme Programming*, Addison-Wesley The XP Series, 2002.

[Davenport, 1998] Davenport, T.H., D.W. DeLong, M.C. Beers, "Successful Knowledge Management Projects," *Sloan Management Rev.*, vol. 39, no. 2, (Winter 1998) pp. 43–57.

[DeMarco, 1987] DeMarco, Tom, Timothy Lister, *Peopleware: Productive Projects and Teams*, Dorset House Publishing Co., 1987.

[DeMarco, 1999] DeMarco, Tom, Timothy Lister, *Peopleware: Productive Projects and Teams, Second Edition*, Dorset House Publishing Co., 1999.

[EA, 2003] *Enterprise Architect v3.61*, Online Help, Sparx Systems, 2003.

[Etkin, 1989] Etkin, Jorge, Leonardo Schvarstein, *Identidad de las Organizaciones: Invariancia y Cambio*, Editorial Paidós, 1989.

[Etkin, 2000] Etkin, Jorge, *Política, Gobierno y Gerencia de las Organizaciones*, Prentice Hall, 2000.

[Fowler, 2000] Fowler, Martin, *The New Methodology*, http://www.martinfowler.com/articles/newMethodology.html, ThoughtWorks Inc., July 2000.

[Fowler, 2001] Fowler, Martin, Cara Taber, *Planning and Running an XP Iteration*, http://www.martinfowler.com/articles/planningXpIteration.html, ThoughtWorks Inc., January 2001.

[Fowler, 2002] Fowler, Martin, Matthew Foemmel, *Continuous Integration*, http://www.martinfowler.com/articles/continuousIntegration.html, ThoughtWorks Inc., 2002.

[Gamma, 1995] Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[Glass, 2002] Glass, Robert L., *Facts and Fallacies of Software Engineering*, Addison-Wesley, 2002.

[Highsmith, 1999] Highsmith, Jim, *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, Dorset House, 1999.

[Highsmith, 2000] Highsmith, Jim, "Retiring Lifecycle Dinosaurs," *Software Testing & Quality Engineering (STQE)*, (July/August 2000) pp. 22-28.

[Highsmith, 2002] Highsmith, Jim, *Agile Software Development Ecosystems*, Addison-Wesley, The Agile Software Development Series, 2002.

[Jacobson, 1999] Jacobson, Ivar, Grady Booch, James Rumbaugh, *The Unified Software Development Process*, Addison-Wesley Object Technology Series, 1999.

[Jeffries, 2001] Jeffries, Ron, Ann Anderson, Chet Hendrickson, *Extreme Programming Installed*, Addison-Wesley The XP Series, 2001.

[Jones, 1997] Jones, Capers, *Programming Languages Table*, Release 8.2, Software Productivity Research, March 1996.

[JUnit, 2001] Beck, Kent, Erich Gamma, *JUnit*, http://www.junit.org/, 2001.

[Kruchten, 2000] Kruchten, Philippe, *The Rational Unified Process: An Introduction, Second Edition*, Addison-Wesley Object Technology Series, 2000.

[Kruchten, 2003] Kruchten, Philippe, Per Kroll, *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Addison-Wesley Object Technology Series, 2003.

[Kulak, 2003] Kulak, Daryl, Eamonn Guiney, *Use Cases: Requirements in Context, Second Edition*, Addison-Wesley, 2003.

[Larman, 2002] Larman, Craig, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design, Second Edition*, Prentice Hall, 2002.

[Larman, 2003] Larman, Craig, *Agile and Iterative Development: A Manager's Guide*, Addison-Wesley Agile Software Development Series, 2003.

[Leffingwell, 2001] Leffingwell, Dean, Don Widrig, *Managing Software Requirements*, Addison-Wesley Object Technology Series, 2001.

[Manifesto, 2001] *Manifesto for Agile Software Development*, http://www.agilealliance.org/, 2001.

[Martin, 1991] Martin, J., *Rapid Application Development*, Macmillan Inc., New York, 1991.

[McBreen, 2002] McBreen, Pete, *Questioning Extreme Programming*, Addison-Wesley The XP Series, 2002.

[McConnell, 1996] McConnell, Steve, *Rapid Development: Taming Wild Software Schedules*, Microsoft Press, 1996.

[McConnell, 2003] McConnell, Steve, *Professional Software Development: Shorter Schedules, Higher Quality Products, More Successful Projects, Enhanced Careers*, Addison Wesley, 2003.

[Nunes, 1999] Nunes, Nuno Jardim, João Falcão e Cunha, *A Bridge too Far: The WISDOM Approach*, ECOOP'99 Workshop on Interactive System Design and Object

Models, Universidade da Madeira, Faculdade de Engenharia da Universidade do Porto, 1999.

[PMBOK, 2000] *Project Management Body of Knowledge*, http://www.pmi.org/, Project Management Institute Inc., 2000.

[Poppendieck, 2003] Poppendieck, Mary, Tom Poppendieck, *Lean Software Development: An Agile Toolkit*, Addison-Wesley, The Agile Software Development Series, 2003.

[Ribu, 2001] Ribu, Kirsten, *Estimating Object-Oriented Software Projects with Use Cases*, Master of Science Thesis, University of Oslo, Department of Informatics, November 2001.

[Royce, 1970] Royce, Winston, *Managing the Development of Large Software Systems*, Proceedings of IEEE Westcon, 1970.

[Royce, 1998] Royce, Walker, *Software Project Management: A Unified Framework*, Addison-Wesley Object Technology Series, 1998.

[RUP, 2002] *Rational Unified Process® Version 2002.05.00*, Copyright © Rational Software Corporation, All Rights Reserved.

[Schenone, 2002] Schenone, Marcelo, *AgEnD: Agility Enhanced Development*, Proceedings, JAIIO 31, ASSE, Agosto 2002.

[Schwaber, 2001] Schwaber, Ken, Mike Beedle, *Agile Software Development with Scrum*, Prentice Hall, 2001.

[Sinan, 1998] Sinan Si Alhir, *UML in a Nutshell*, O'Reilly & Associates, 1998.

[Smith, 2001] Smith, John, *A Comparison of RUP® and XP*, Rational Software White Paper, 2001.

[Standish, 1994] The Standish Group, "The CHAOS Report," The Standish Group International, 1994.

[Standish, 1996] The Standish Group, "Unfinished Voyages: A Follow-Up to The CHAOS Report," The Standish Group International, 1996.

[Standish, 1999] The Standish Group, "CHAOS: A Recipe for Success," The Standish Group International, 1999.

[Stapleton, 1997] Stapleton, Jennifer, *DSDM Dynamic Systems Development Method*, Addison-Wesley, 1997.

[Sun, 2003] *Architecting and Designing J2EE™ Applications*, Copyright 2003 Sun Microsystems Inc., 2003.

[SWEBOK, 2001] *Software Engineering Body of Knowledge*, http://www.swebok.org/, 2001.

[Wainstein, 2000] Wainstein, Martín, *Intervenciones con Individuos, Parejas, Familias y Organizaciones*, EUDEBA, Universidad de Buenos Aires, 2000.

[Weinberg, 1998] Weinberg, Gerald M., *The Psychology of Computer Programming*, Silver Edition, Dorset House, 1998.

[Williams, 2000] Williams, Laurie, Robert R. Kessler, Ward Cunningham, Ron Jeffries, "Strengthening the Case for Pair Programming," *IEEE Software*, Vol. 17, No. 4 (July/August 2000) pp. 19-25.

[Williams, 2002] Williams, Laurie, Robert R. Kessler, *Pair Programming Illuminated*, Addison-Wesley, 2002.

Links en Internet sobre Metodologías Ágiles

AgEnD	Marcelo Schenone: http://www.agend.com.ar/
Agile Alliance	Agile Alliance: http://www.agilealliance.com/
Adaptive Software Development	Jim Highsmith: http://www.asd.com/
Crystal	Alistair Cockburn: http://crystalmethodologies.org/
DSDM	DSDM Consortium: http://www.dsdm.org/
dX	Robert Martin: http://www.objectmentor.com/publications/RUPvsXP.pdf
Extreme Modeling	Scott Ambler: http://www.extreme-modeling.com/
FDD	Peter Coad, Eric Lefebvre, Jeff De Luca: http://www.cs.jhu.edu/~scott/oos/software/togetherj/help/Users-Guide/Feature_Driven_Development.htm Peter Coad: http://www.togethercommunity.com/coad-letter/Coad-Letter-0070.html
SCRUM	Ken Schwaber: http://www.controlchaos.com/ Jeff Sutherland: http://jeffsutherland.com/scrum/
Usage-Centered Design	Larry Constantine: http://www.foruse.com/
XBreed	Mike Beedle: http://www.xbreed.net/
XP	Ron Jeffries: http://www.xprogramming.com/ Robert Martin: http://www.objectmentor.com/ Don Wells: http://www.extremeprogramming.org/ Jim Highsmith on “Extreme Programming”: http://www.cutter.com/ead/ead0002.html

Descripción del Proyecto
Documento de Visión

<Versión>

<Autor>

Historia de Revisión

Fecha	Versión	Descripción	Autor

Tabla de Contenidos

1. Introducción

[Descripción del documento y del contexto del proyecto de desarrollo. Se identifica al Cliente y el propósito detrás de la construcción del sistema.]

Referencias

[Indicaremos todos los artefactos relacionados con este documento.]

2. Posicionamiento

Situación Actual

[Descripción del problema actual que se presenta y la forma en que el sistema beneficiará a los usuarios mediante una solución dada.]

3. Stakeholders y Usuarios

Stakeholders identificados

[Tabla que muestra a los stakeholders involucrados en el proyecto. Se da una breve descripción de cada uno y se analiza el impacto de este en la solución.]

Usuarios identificados

[Tabla que muestra a los usuarios que tendrá el sistema. Se da una breve descripción de cada uno y se analiza las responsabilidades en relación al sistema en construcción.]

4. Características del Producto

[A continuación se detallan las características del producto en forma de features de alto nivel. Estos identifican las capacidades que tendrá el sistema para cubrir las necesidades de los usuarios.]

Features de Alto Nivel

[Feature 1.]

[Feature 2.]

...

[Feature X.]

5. Otros Requerimientos

[A continuación se detallan las características del producto en forma de requerimientos no funcionales de alto nivel, restricciones a ser impuestas a la solución y reglas del negocio que determinarán la descripción de la arquitectura candidata resultante.]

Requerimientos No Funcionales

[Requerimiento No Funcional 1.]

[Requerimiento No Funcional 2.]

...

[Requerimiento No Funcional X.]

Restricciones

[Restricción 1.]

[Restricción 2.]

...

[Restricción X.]

Reglas del Negocio Clave

[Regla del Negocio 1.]

[Regla del Negocio 2.]

...

[Regla del Negocio X.]

Descripción del Proyecto
Lista de Riesgos

<Versión>

<Autor>

Historia de Revisión

Fecha	Versión	Descripción	Autor

Tabla de Contenidos

1. Introducción

Propósito

[Breve comentario del propósito de este documento. Se puede tomar la descripción de AgEnD.]

2. Riesgos

[A continuación se enumerarán todos los riesgos que presenta el proyecto. En dicha lista deben figurar tanto los riesgos de management identificados por el Líder de Proyecto, como los riesgos más técnicos identificados por el Arquitecto.]

<<Identificación del Primer Riesgo>>

- **Descripción**

  [Breve descripción del riesgo.]

- **Criticidad**

  [Indicar el nivel de criticidad en caso de que el riesgo se materialice. El rango puede estar en una escala de alta, media o baja.]

- **Probabilidad de Ocurrencia**

  [Indicar la probabilidad de que ocurra el riesgo en el proyecto. El rango puede estar en una escala de alta, media o baja.]

- **Impacto**

  [El impacto está dado por el producto de la criticidad y la probabilidad de ocurrencia. En base a esto los riesgos serán priorizados y monitoreados por el Líder de Proyecto y el Arquitecto. Por ejemplo, un riesgo con criticidad alta y probabilidad de ocurrencia alta deberá ser monitoreado frecuentemente.]

- **Estrategia de Mitigación**

  [Indicar la estrategia a ser utilizada para minimizar el impacto del riesgo en caso que este se materialice.]

- **Plan de Contingencia**

  [Indicar el plan de contingencia que será tenido en cuenta en caso que la estrategia de mitigación no tenga éxito.]

<<Identificación del Segundo Riesgo>>

...

Descripción del Proyecto
Código y Nombre del Caso de Uso

<Versión>

<Autor>

Historia de Revisión

Fecha	Versión	Descripción	Autor

Tabla de Contenidos

1. Breve Descripción

[Breve descripción del propósito del caso de uso. Recordar que el mismo debe dar algún valor a algún actor del sistema.]

2. Actores

[Enumerar los actores que participan en este caso de uso.]

3. Flujo de Eventos

[A continuación se describe la secuencia de pasos desde que el actor dispara el caso de uso hasta que la interacción llega a su fin.]

3.1. Flujo Básico

[El flujo básico representa la interacción normal del caso de uso. Es la situación en que todos los pasos se ejecutan normalmente, sin existir excepciones ni errores.]

3.2. Flujos Alternativos

[Los flujos alternativos se dan ante alternativas complejas dentro de la ejecución del flujo las cuales ocurren ante ciertos eventos alternos de la interacción.]

3.3. Flujos de Excepción

[Los flujos de excepción ocurren cuando existen excepciones en la ejecución del flujo básico debida a errores internos de la aplicación o de la interacción del usuario con esta.]

4. Requerimientos Suplementarios

[Indicar aquellos requerimientos suplementarios relacionados únicamente con este caso de uso. Recordar que los requerimientos suplementarios globales van en el SRS]

5. Precondiciones

[Indicar las condiciones que deben darse para que se pueda disparar la ejecución de este caso de uso.]

6. Poscondiciones

[Indicar como quedará el sistema después que se termine la ejecución de este caso de uso.]

7. Puntos de Extensión

[Puntos a partir de los cuales se extiende el flujo básico de este caso de uso. Relacionado con la relación <<extends>> de UML.]

8. Diagramas Asociados

[Colocar aquellos diagramas asociados con este caso de uso. En este punto se puede colocar el modelo de casos de uso afectado a este caso de uso, diagrama de estado, diagrama de actividad, prototipo de GUI, etc.]

9. Supuestos y Dudas

[Dejar constancia de los supuestos que se han tomado en la especificación del casos de uso así como las dudas que el Analista tiene pendiente y deberán ser resueltas con el Cliente.]

Descripción del Proyecto
Documento de Especificación de Requerimientos de Software (SRS)

<Versión>

<Autor>

Historia de Revisión

Fecha	Versión	Descripción	Autor

Tabla de Contenidos

1. Introducción

Propósito

[Breve comentario del propósito de este documento. Se puede tomar la descripción de AgEnD.]

Referencias

[Indicaremos todos los artefactos relacionados con este documento.]

2. Requerimientos Funcionales

Detalle de Requerimientos Funcionales

[A continuación se enumerarán todos aquellos requerimientos de carácter funcional que tengan que ver con aspectos más técnicos relacionados con la solución. Los mismos quedan fuera de las especificaciones de los casos de uso.]

3. Requerimientos Suplementarios

Requerimientos No Funcionales

[A continuación se enumerarán todos aquellos requerimientos no funcionales que son globales a la aplicación. Los mismos refieren a las cualidades sistémicas asociadas con: usabilidad, performance, mantenibilidad, seguridad, tolerancia, etc.]

Restricciones

[A continuación se enumerarán todas las restricciones que se aplican a la solución planteada. Las mismas restringen los grados de libertad al momento de diseñar una solución determinada a un problema.]

Reglas del Negocio

[A continuación se enumerarán todas aquellas reglas del negocio globales a la aplicación. Las mismas tienen que ver con las políticas y/o condiciones que se dan en los procesos del negocio y que deben ser implementadas en la solución.]

<<Logo Empresa>>

Descripción del Proyecto
Descripción de Arquitectura

<Versión>

<Autor>

Historia de Revisión

Fecha	Versión	Descripción	Autor

Tabla de Contenidos

1. Introducción

Propósito

[Breve comentario del propósito de este documento. Se puede tomar la descripción de AgEnD.]

Referencias

[Indicaremos todos los artefactos relacionados con este documento.]

2. Objetivos Arquitectónicos

[Se mencionarán los requerimientos no funcionales y restricciones que guiaron el trabajo del arquitecto. Los mismos estarán basados en lo que fue documentado en el SRS, pero haciendo énfasis sobre como la arquitectura candidata resuelve cada uno de estos.]

3. Vista de Casos de Uso

[Es una descripción de una vista de la arquitectura del software a través de los casos de usos. Representa el conjunto de escenarios de la iteración que describen o representan la funcionalidad requerida para el proyecto.]

4. Vista Lógica

[Es una descripción de una vista de la arquitectura del software a través de la evaluación realizada de los requerimientos funcionales y no funcionales. Representa la organización en paquetes de servicios y subsistemas y la organización de estos subsistemas en layers. También se describen aquí las realizaciones de caso de uso más importantes.]

5. Vista de Despliegue

[La vista de despliegue permitirá describir la distribución física posible de los diferentes nodos de procesamiento describiendo además el modelo de crecimiento de la solución.]

6. Bibliografía

[En este apartado se podrá mencionar la bibliografía de donde se puede obtener más información de los elementos mencionados en este documento.]

Descripción del Proyecto
Caso de Prueba

<Versión>

<Autor>

Historia de Revisión

Fecha	Versión	Descripción	Autor

Tabla de Contenidos

1. Introducción

Propósito

[Breve comentario del propósito de este documento. Se puede tomar la descripción de AgEnD.]

Referencias

[Indicaremos todos los artefactos relacionados con este documento.]

2. Definición de Casos de Prueba

Número Caso de Prueba

[Número del caso de prueba usado para identificación y posteriormente para tener estadísticas.]

Título Caso de Prueba

[Título descriptivo del caso de prueba.]

ID de Caso de Uso

[Identificación del caso de uso. Por ejemplo, podría ser algo como CUXXX Alta de Usuarios.]

Módulo/Componente

[Módulo de la aplicación que esta siendo probado.]

Funcionalidad

[Funcionalidad de la aplicación que se prueba.]

Modo de Prueba

[Manual o automático (mediante algún script o robot).]

Preparación de Configuración

[Preparación de configuración para llevar a cabo la prueba. En este punto colocaremos cualquier infraestructura o configuración o carga de datos que requiera la prueba.]

Procedimiento de la Prueba

[Secuencia de pasos para llevar a cabo la prueba.]

Resultados Esperados

[Resultados esperados por la prueba.]

Descripción del Proyecto
Nota de Entrega

<Versión>

<Autor>

Historia de Revisión

Fecha	Versión	Descripción	Autor

Tabla de Contenidos

1. Introducción

Propósito

[Breve comentario del propósito de este documento. Se puede tomar la descripción de AgEnD.]

Referencias

[Indicaremos todos los artefactos relacionados con este documento.]

2. Información de la Entrega

Overview

[Breve descripción del contenido de la entrega que se lleva a cabo. Se enumeran todos los artefactos que la misma incluye, identificando el número de versión donde corresponda.]

3. Nuevos Features

Lista de Nuevos Features

[Listado de los features que están contenidos en la versión que se entrega al Cliente. Orientado a entregables de implementación.]

Features

[Feature 1.]

[Feature 2.]

...

[Feature X.]

4. Bugs Conocidos o Limitaciones

Lista de Bugs Conocidos

[Listado de los bugs conocidos que no han podido ser corregidos para la corriente entrega. Los mismos serán identificados de forma que el Cliente los conozca y no necesite reportarlos nuevamente al equipo.]

Bugs

[Bug 1.]

[Bug 2.]

...

[Bug X.]

Lista de Limitaciones

[Listado de las limitaciones de la versión entregada. Se detallarán aquellas cuestiones que quedaron fuera de la implementación corriente.]

Limitaciones

[Limitación 1.]

[Limitación 2.]

...

[Limitación X.]