LAS METODOLOGÍAS AGILES

Varias metodologías encajan bajo el estandarte de ágil. Mientras todas ellas comparten muchas características, también hay algunas diferencias significativas. No puedo resaltar todos los puntos en este estudio breve, pero por lo menos puedo apuntar a algunos lugares donde buscar. Tampoco puedo hablar con experiencia significativa sobre la mayoría de ellas. He trabajado bastante en XP, y he visto el RUP en muchas capas, pero para la mayoría de los otros mi conocimiento no es el más adecuado.

DE NADA A MONUMENTAL A AGIL

Con mucho el desarrollo de software es una actividad caótica, frecuentemente caracterizada por la frase "codifica y corrige". El software se escribe con un mínimo un plan subyacente, y el diseño del sistema se adoquina con muchas decisiones a corto plazo. Esto realmente funciona muy bien si el sistema es pequeño, pero conforme el sistema crece llega a ser cada vez más difícil agregar nuevos aspectos al mismo. Además los bugs llegan a ser cada vez más frecuentes y más difíciles de corregir. La seña típica de tal sistema es una larga fase de pruebas después de que el sistema ha sido "completado". Tal fase larga de pruebas hace estragos con los planes de pruebas y depurado llegando a ser imposible de poner en el programa de trabajo.

Hemos vivido con este estilo de desarrollo por mucho tiempo, pero también hemos tenido una alternativa desde hace mucho: Metodología. Las metodologías imponen un proceso disciplinado sobre el desarrollo de software con el fin de hacerlo más predecible y eficiente. Lo hacen desarrollando un proceso detallado con un fuerte énfasis en planificar inspirado por otras disciplinas de la ingeniería.

Las metodologías ingenieriles han estado presentes durante mucho tiempo. No se han distinguido precisamente por ser muy exitosas. Aún menos por su popularidad. La crítica más frecuente a estas metodologías es que son burocráticas. Hay tanto que hacer para seguir la metodología que el ritmo entero del desarrollo se retarda.

Como una reacción a estas metodologías, un nuevo grupo de metodologías ha surgido en los últimos años. Durante algún tiempo se conocían como las metodologías ligeras, pero el término aceptado ahora es metodologías ágiles. Para mucha gente el encanto de estas **metodologías ágiles** es su reacción a la burocracia de las metodologías monumentales. **Estos nuevos métodos buscan un justo medio entre ningún**

proceso y demasiado proceso, proporcionando simplemente suficiente proceso para que el esfuerzo valga la pena.

El resultado de todos esto es que los métodos ágiles cambian significativamente algunos de los énfasis de los métodos ingenieriles. La diferencia inmediata es que son menos orientados al documento, exigiendo una cantidad más pequeña de documentación para una tarea dada. En muchas maneras son más bien orientados al código: siguiendo un camino que dice que la parte importante de la documentación es el código fuente.

- *Los métodos ágiles son adaptables en lugar de predictivos*. Los métodos ingenieriles tienden a intentar planear una parte grande del proceso del software en gran detalle para un plazo grande de tiempo, esto funciona bien hasta que las cosas cambian. Así que su naturaleza es resistirse al cambio. Para los métodos ágiles, no obstante, el cambio es bienvenido. Intentan ser procesos que se adaptan y crecen en el cambio, incluso al punto de cambiarse ellos mismos.
- *Los métodos ágiles son orientados a la gente y no orientados al proceso*. La meta de los métodos ingenieriles es definir un proceso que funcionará bien con cualquiera que lo use. Los métodos ágiles afirman que ningún proceso podrá nunca maquillar las habilidades del equipo de desarrollo, de modo que el papel del proceso es apoyar al equipo de desarrollo en su trabajo. Explícitamente puntualizan el trabajar a favor de la naturaleza humana en lugar de en su contra y enfatizan que el desarrollo de software debe ser una actividad agradable.

SEPARACIÓN DE DISEÑO Y CONSTRUCCIÓN

La inspiración usual para las metodologías han sido disciplinas como las ingenierías civil o mecánica. Tales disciplinas enfatizan que hay que planear antes de construir. Los ingenieros trabajan sobre una serie de esquemas que indican precisamente qué hay que construir y como deben juntarse estas cosas. Muchas decisiones de diseño, como la manera de controlar la carga sobre un puente, se toman conforme los dibujos se producen. Los dibujos se entregan entonces a un grupo diferente, a menudo una compañía diferente, para ser construidos. Se supone que el proceso de la construcción seguirá los dibujos. En la práctica los constructores se encuentran con algunos problemas, pero éstos son normalmente poco importantes.

Como los dibujos especifican las piezas y cómo deben unirse, actúan como los fundamentos de un plan de construcción detallado. Dicho plan define las tareas que necesitan hacerse y las dependencias que existen entre estas tareas. Esto permite un plan de trabajo y un presupuesto de construcción razonablemente predecibles. También dice en detalle cómo deben hacer su trabajo las personas que participan en la construcción. Esto permite que la construcción requiera menos pericia intelectual, aunque se necesita a menudo mucha habilidad manual.

Así que lo que vemos aquí son dos actividades fundamentalmente diferentes. El diseño, qué es difícil de predecir y requiere personal caro y creativo, y la construcción que es más fácil de predecir. Una vez que tenemos el diseño, podemos planear la construcción. Una vez que tenemos el plan de construcción, podemos ocuparnos de la construcción de una manera más predecible. En ingeniería civil la construcción es mucho más costosa y tardada que el diseño y la planeación.

Así el acercamiento de muchas metodologías es: queremos un plan de trabajo predecible que pueda usar gente del más bajo nivel. Para hacerlo debemos separar el plan de la construcción. Por consiguiente necesitamos entender cómo hacer el diseño de software de modo que la construcción pueda ser sencilla una vez que el plan esté hecho.

LA IMPREDECIBILIDAD DE LOS REQUISITOS

Hay un dicho que oigo en cada proyecto problemático con que me he encontrado. Los desarrolladores vienen y me dicen "el problema con este proyecto es que los requisitos cambian todo el tiempo". Lo que yo encuentro sorprendente sobre esta situación es que sorprenda a cualquiera. En el negocio de construcción de software los cambios en los requisitos son la norma, la pregunta es qué hacemos al respecto.

Una forma es tratar los requisitos cambiantes como el resultado de una pobre ingeniería de requisitos. La idea detrás de la ingeniería de requisitos es conseguir un cuadro totalmente entendido de los requisitos antes de empezar a construir el software, conseguir la firma del cliente sobre estos requisitos, y entonces preparar procedimientos que limiten los cambios de requisitos después de la firma.

Un problema con esto es que simplemente tratar de entender las opciones para los requisitos es duro. Es aun más duro porque la organización del desarrollo normalmente no proporciona la información del costo en los requisitos.

Usted termina en la situación dónde quisiera tener un quemacocos en su automóvil, pero el vendedor no puede decirle si agrega 10 al costo del automóvil, o 10,000. Sin una buena idea del costo, ¿cómo puede usted decidir si quiere pagar por ese quemacocos?

La estimación es difícil por muchas razones. En parte porque el desarrollo de software es una actividad de diseño, difícil de planear y costar. En parte porque los materiales básicos cambian rápidamente. En parte por lo mucho que depende de los individuos involucrados, y los individuos son difíciles de predecir y cuantificar.

La naturaleza intangible del software también afecta. Es muy difícil saber qué valor aporta un rasgo de software hasta que se usa en realidad. Sólo cuando se usa realmente una versión temprana de algún software se empieza a entender qué rasgos son valiosos y cuales no.

Esto lleva al punto irónico de que es de esperarse que los requisitos sean cambiables. Después de todo se supone que el software es "suave". Así no sólo son cambiables los requisitos, sino que deben de serlo. Toma mucha energía conseguir que los clientes de software corrijan los requisitos. Es aun peor si ellos han estando alguna vez en desarrollo de software, porque entonces "saben" que el software es fácil de cambiar.

Pero aun cuando se pudiera controlar todo esto y realmente se pudiera conseguir un conjunto exacto y estable de requisitos, probablemente aún no estamos a salvo. En la economía de hoy las fuerzas de negocios fundamentales cambian el valor de los rasgos de software demasiado rápidamente. El que podría ser un buen conjunto de requisitos ahora, no será tan bueno en seis meses. Aun cuando el cliente pueda corregir sus requisitos, el mundo de negocios no va a detenerse por él. Y muchos cambios en el mundo de negocios son completamente imprevisibles: cualquiera que diga otra cosa o está mintiendo, o ya ha hecho mil millones en la bolsa de valores.

PONIENDO A LA GENTE PRIMERO

No es fácil ejecutar un proceso adaptable. En particular requiere un equipo muy eficaz de desarrolladores. El equipo necesita ser eficaz tanto en la calidad de los individuos como en la manera en que funcionan juntos en equipo. Hay también una sinergia interesante: no sólo la adaptabilidad requiere un equipo fuerte, la mayoría de los buenos desarrolladores prefieren un proceso adaptable.

LOS PROGRAMADORES SON PROFESIONALES RESPONSABLES

Una parte clave de la noción Taylorista es que la gente que hace el trabajo no es la mejor gente para entender la mejor manera de hacer el trabajo. En una fábrica esto puede ser verdad por varias razones. En parte porque la mayoría de los obreros no son las personas más inteligentes o creativas, en parte porque hay una tensión entre la gerencia y los obreros en que la gerencia gana más dinero y los obreros menos.

La historia reciente nos demuestra cada vez más lo falso que es esto para el desarrollo de software. A las personas brillantes y capaces les atrae cada vez más el desarrollo de software, tanto por su ostentación como por ganancias potencialmente mayores. (Ambas razones me tentaron a salir de la ingeniería electrónica.) Cosas tales como opciones accionarias afirman el interés de los programadores en la compañía.

(Aquí bien puede haber un efecto generacional. Una evidencia anecdótica me hace preguntarme si más gente brillante se han aventurado en el desarrollo de software en los últimos diez años. En ese caso ésta sería una razón para ese culto a la juventud en el negocio de la computación, como en la mayoría de los cultos se necesita un grano de verdad en él.)

Cuando se quiere contratar y retener a gente capaz, hay que reconocer que son profesionales competentes. Como tales son los mejores para decidir cómo dirigir su trabajo técnico. La noción Taylorista de un departamento de planificación separado que decide cómo hacer las cosas sólo funciona si los planificadores entienden mejor cómo hacer bien el trabajo que aquéllos que lo hacen. Si usted tiene personas brillantes, motivadas que hacen bien el trabajo entonces eso no se sostiene.

MANEJANDO UN PROCESO ORIENTADO A LA GENTE

La orientación a la gente se manifiesta de varias maneras diferentes en los procesos ágiles, lo que lleva a efectos diferentes, no todos consistentes.

Uno de los elementos clave es la aceptación de un proceso en lugar de la imposición de un proceso. A menudo los procesos de software se imponen desde la gerencia. Como tales se les resiste a menudo, particularmente cuando la gerencia ha estado fuera del desarrollo activo un buen

tiempo. Aceptar un proceso requiere compromiso, y como tal se necesita el involucramiento activo de todo el equipo.

Esto termina con el resultado interesante de que sólo los desarrolladores puede escoger seguir un proceso adaptable. Esto es particularmente cierto para la XP, que requiere mucha disciplina para ejecutarse. Aquí es donde Cristal es un complemento efectivo ya que apunta a una disciplina mínima.

Otro punto es que los desarrolladores deben poder tomar todas las decisiones técnicas. XP llega al corazón de esto cuando en su proceso de planeación establece que sólo los desarrolladores pueden estimar cuánto tiempo tomará hacer un trabajo.

Tal liderazgo técnico es un gran cambio para muchas personas en posiciones gerenciales. Tal acercamiento requiere compartir una responsabilidad donde desarrolladores y gerencia tienen un mismo lugar en la dirección del proyecto. Nótese que dije *igual*. La gerencia aun juega un papel, pero reconoce la pericia de los desarrolladores.

Una razón importante para esto es el ritmo del cambio de tecnología en nuestra industria. Después de unos años el conocimiento técnico se vuelve obsoleto. Esta vida media de habilidades técnicas no tiene parangón en cualquier otra industria. Incluso los técnicos tienen que reconocer que entrar a la gerencia significa que sus habilidades técnicas se marchitarán rápidamente. Los exdesarrolladores necesitan reconocer que sus habilidades técnicas desaparecerán rápidamente y necesitan confiar y depender en los desarrolladores actuales.

SCRUM

Scrum ha estado durante algún tiempo en los círculos orientados a objetos, aunque confesaré que yo no estoy muy al tanto de su historia o desarrollo. De nuevo se enfoca en el hecho de que procesos definidos y repetibles sólo funcionan para atacar problemas definidos y repetibles con gente definida y repetible en ambientes definidos y repetibles.

Scrum divide un proyecto en iteraciones (que ellos llaman carreras cortas) de 30 días. Antes de que comience una carrera se define la funcionalidad requerida para esa carrera y entonces se deja al equipo para que la entregue. El punto es estabilizar los requisitos durante la carrera.

Sin embargo la gerencia no se desentiende durante la carrera corta. Todos los días el equipo sostiene una junta corta (quince minutos), llamada scrum, dónde el equipo discurre lo que hará al día siguiente. En particular muestran a los bloques de la gerencia: los impedimentos para progresar que se atraviesan y que la gerencia debe resolver. También informan lo que se ha hecho para que la gerencia tenga una actualización diaria de dónde va el proyecto.

La literatura de Scrum se enfoca principalmente en la planeación iterativa y el seguimiento del proceso. Es muy cercana a las otras metodologías agiles en muchos aspectos y debe funcionar bien con las prácticas de código de la XP.

DESARROLLO MANEJADO POR RASGOS

El Desarrollo Manejado por Rasgos (FDD por sus siglas en inglés) fue desarrollado por Jeff De Luca y el viejo gurú de la OO Peter Coad. Como las otras metodologías adaptables, se enfoca en iteraciones cortas que entregan funcionalidad tangible. En el caso del FDD las iteraciones duran dos semanas.

El FDD tiene cinco procesos. Los primeros tres se hacen al principio del proyecto.

- Desarrollar un Modelo Global
- Construir una Lista de los Rasgos
- Planear por Rasgo
- Diseñar por Rasgo
- Construir por Rasgo

Los últimos dos se hacen en cada iteración. Cada proceso se divide en tareas y se da un criterio de comprobación.

Los desarrolladores entran en dos tipos: dueños de clases y programadores jefe. Los programadores jefe son los desarrolladores más experimentados. A ellos se les asignan rasgos a construir. Sin embargo ellos no los construyen solos. Solo identifican qué clases se involucran en la implantación de un rasgo y juntan a los dueños de dichas clases para que formen un equipo para desarrollar ese rasgo. El programador jefe actúa como el coordinador, diseñador líder y mentor mientras los dueños de clases hacen gran parte de la codificación del rasgo.

Hasta recientemente, la documentación sobre FDD era muy escasa. Finalmente hay un libro completo sobre FDD. Jeff De Luca, el inventor primario, ya tiene un portal FDD con artículos, blogs y foros de discusión. La descripción

original estaba en el libro UML in Color de Peter Coad et al. Su compañía, TogetherSoft, también da consultoría y entrenamiento en FDD.

Metodología De Trabajo

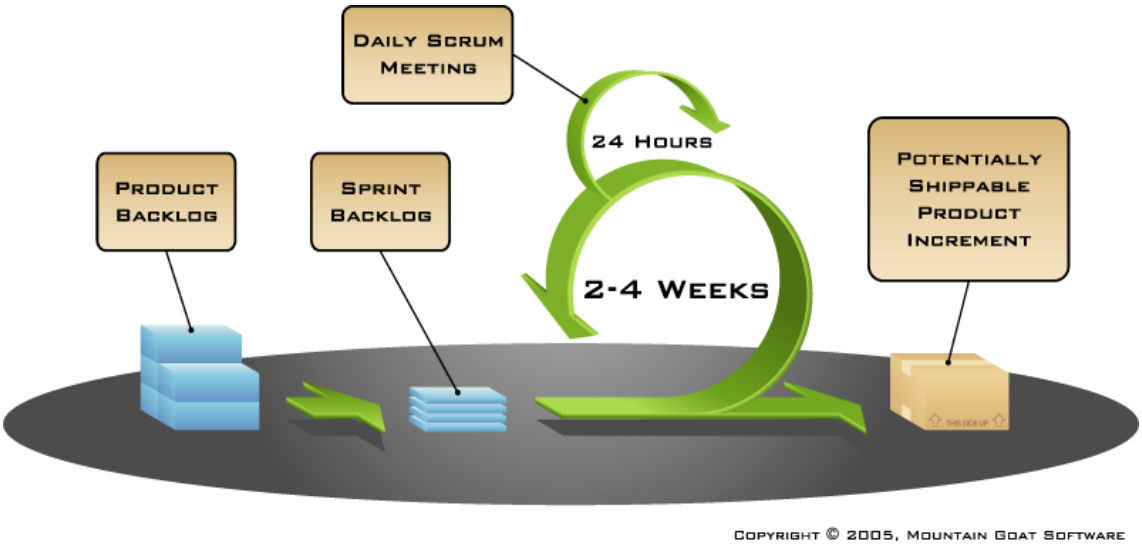
Equipos de entre 6 y 10 personas revisan los requisitos, la tecnología disponible y evalúan los conocimientos para Colectivamente determinar como **incrementar la uncionalidad.**

* Reuniones diarias, antes de empezar a trabajar,

  con una duración máxima de 4 hrs.

* Se llevan a cabo hasta que el proyecto este listo

  para ser puesto en producción o ser lanzado al

  mercado.

EL FLUJO SCRUM