Modelos ágiles de desarrollo

Alejandro Teruel

5 de mayo 2013

Contenido: Características de los modelos ágiles. Ventajas, desventajas y riesgos resaltantes. Iteraciones, fases y dinámica de desarrollo.

Brian Cantrill: Software has often been compared with civil engineering, but I'm really sick of people describing software as being like a bridge. What do you think the analog for software is?

Arthur Whitney: Poetry.

Brian Cantrill: Poetry captures the aesthetics, but not the precision.

Arthur Whitney: I don't know, maybe it does.

Brian Cantrill: *A Conversation with Arthur Whitney.* 1 Febrero 2009. http://queue.acm.org/detail.cfm?id=1531242 Consultado 4 de mayo 2013

Manifiesto por el Desarrollo Ágil de Software

En el año 2001 un grupo de desarrolladores de software que incluyeron a Kent Beck, Alistair Cockburn, Ward Cunningham, Martin Fowler, Brian Marick, Steve Mellor y Jeff Sutherland firmaron y publicaron un "manifiesto" a favor de lo que denominaron como el *desarrollo ágil de software.*

> *Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar:*
>
> ** Individuos e interacciones sobre procesos y herramientas*
>
> ** Software funcionando sobre documentación extensiva*
>
> ** Colaboración con el cliente sobre negociación contractual*
>
> ** Respuesta ante el cambio sobre seguir un plan*
>
> *Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda.*
>
> http://agilemanifesto.org/iso/es/

El paradigma dominante para el momento era que cualquier proceso serio y profesional de ingeniería de software debía tener por modelo a las ingenierías clásicas como la Ingeniería Civil. Ya para 1986 dos computistas entrevistaron a un ingeniero de puentes con el objetivo de mirar más de cerca esa analogía[1]. La analogía sugiere un modelo de desarrollo en el que la construcción es precedido por fases muy completas de análisis y diseño que culminan en un detallado documento-plano de especificaciones que se entregaba al contratista de la construcción.

1 La entrevista fue hecha a Gerald Fox por Alfred Spector y David Gifford y publicada como "A Computer Science Perspective on Bridge Design" en Communications of the ACM, Abril 1986. http://portal.acm.org/citation.cfm?id=5684.6327&coll=ACM&dl=ACM&idx=J79&part=magazine&WantType=Magazines&title=Communications%20of%20the%20ACM&CFID=47665715&CFTOKEN=13076261

En el fondo ese paradigma estaba fundamentado en una *visión burocrática* del trabajo. La propia palabra *burocracia* proviene de la yuxtaposición de una palabra en Francés *bureau* (oficina o escritorio) y la palabra griega *kratos* (gobierno). Así como *democracia* es el gobierno por el pueblo, *burocracia* sería entonces el gobierno por parte de los administradores o quizás más precisamente, el gobierno desde el *cargo*. Los fundamentos conceptuales de la burocracia son:

- Cada individuo ejerce un *cargo,* un rol en que se precisan y delimitan las responsabilidades y la estructura de las comunicaciones. El individuo en un cargo tiene unas tareas muy precisas, un conjunto de *procedimientos* que debe seguir y un conjunto de *normas* que debe cumplir. Puede delegar ciertas tareas y recopilar información de sus subalternos, así como solicitar instrucciones o decisiones a sus superiores cuando el asunto que trata rebasa las competencias del cargo.
- Ningún individuo es imprescindible. Los individuos son componentes reemplazables con relativo poco esfuerzo (un poco de entrenamiento, ascenso) en la maquinaria que conforma la burocracia.
- Toda tarea se verifica y puede auditarse.

Según la visión burocrática, el cumplimiento de los objetivos de una institución es demasiado importante para que depende de un individuo, o de un grupo de individuos en particular. De allí que se debe estar preparado, en todo momento para reemplazar un individuo que se enferma, renuncia, se jubila o muere. Esta preparación sólo se puede lograr si toda decisión importante es documentada, si todo procedimiento es explícitamente cumplido y si se respetan las competencias asignados a los cargos.

De esta manera la *burocratización* de la ingeniería del software conlleva a la creación de cargos específicos de responsabilidad limitada: gerente de proyecto, analista, diseñador de arquitectura, diseñador de detalle, programador senior, programador junior, responsable de herramientas de soporte, especialista en bases de datos, planificador de pruebas, *tester*, escritor técnico y así sucesivamente. Todo se documenta y todo se comunica por escrito siguiendo canales, procedimientos y normas preestablecidas. Esta visión de la ingeniería del software se ha descrito como *dirigidos por planes* y de peso pesado (*heavy-weight)*, haciendo referencia, en este último caso, tanto al peso del *overhead* como a su supuesta idoneidad para hacerse cargo de proyectos complejos y de gran envergadura.

Muchos de los proyectos de desarrollo burocratizados reportaron que el análisis y diseño del software se llevaba alrededor del 40% del esfuerzo y trabajo, la validación y verificación otro 40% dejando entre 10 y 20% para la programación en proyectos que podían durar lustros. El rol que se le quedaba al programador era equiparable al rol del obrero de la construcción, un rol que resultaba poco estimulante por cuanto su tarea estaba limitado por el diseño detallado que le era entregado, por estándares de programación, presiones para entregar código contra reloj y por una feroz actividad adversarial por parte de los responsables del aseguramiento de calidad.

Este modelo se hacía cada vez más insatisfactorio sobre todo ante las necesidades cada vez más apremiantes de clientes insertos en un ambiente altamente competitivo caracterizado adicionalmente la incertidumbre y la volatilidad de los requerimientos: para la empresa cliente el software desarrollado por métodos tradicionales frecuentemente era francamente insatisfactorio en su funcionalidad, y se entregaba demasiado tarde a costos que consideraba excesivos. Para los clientes los desarrolladores no parecían entender las necesidades reales del negocio.

En este sentido el modelo representado por el manifiesto por el desarrollo ágil de software, aunque

apalancado por modelos más iterativos, fue visto, inicialmente como una irrupción protestaria e irreverente de *hackers*. Analicemos cada elemento de la manifestación:

- *[Valoramos] Software funcionando sobre documentación extensiva*

  La esencia del desarrollo de software es *software que funciona*, no documentación, por ende el foco de la atención debe estar en el software, no en su documentación.

- *[Valoramos] Individuos e interacciones sobre procesos y herramientas*

  Esta frase deliberadamente busca romper con la visión conservadora y mecanicista del un modelo burocratizado de la ingeniería de software. El nuevo modelo busca apoyarse sobre la equipos altamente motivados que estimulan la contribución de sus miembros, el estímulo de las comunicaciones y sobre el trabajo conjunto y no fragmentado. El individuo no está confinado a su cargo sino está potenciado por sus competencias y sus compañeros de equipo. Unido a la frase anterior, representó para muchos el reconocimiento central del programador que trasciende su limitado rol y también contribuye al diseño y al aseguramiento de calidad.

- *[Valoramos] Colaboración con el cliente sobre negociación contractual*

  Para la visión burocratizada el cliente era externo al desarrollo, especificaba lo que quería y se esperaba que se desapariera hasta que se le entregara lo que había pedido en el contrato. El contrato era un documento clave, hecho desde la desconfianza de dos adversarios naturales. El desarrollador cuidaba celosamente su parcela y especificaba en el contrato cómo y cuándo podía el cliente introducir cambios a los requerimientos, reservándose la opción de tener que renegociar el contrato debido a la introducción de esos cambios o de posponer tales cambios para una futura versión del software. En la versión ágil se busca eliminar la desconfianza, fomentando una relación *ganar-ganar*, abriendo el proceso de desarrollo al cliente y más que invitándolo a participar, exigiendo que lo hiciera para obtener sus aclaratorias y su retroalimentación sobre el proyecto lo antes posible.

- *[Valoramos]Respuesta ante el cambio sobre seguir un plan*

  El modelo ágil reconoce claramente la realidad de la volatilidad de los requerimientos y reconoce que lo fundamental para el cliente es obtener algo que le sirva, que tenga valor para su negocio.

Para algunos, el modelo ágil era anárquico, "anti-metodologías". Pero si se lee con cuidado el manifiesto ("*...aunque valoramos los elementos de la derecha, valoramos más los de la izquierda"*) no se trataba de eliminar toda documentación, contratos, análisis o diseño. Jim Highsmith, otro de los firmantes del manifiesto lo aclaró:

> "... muchos de nosotros queremos retaurarle credibilidad a la palabra metodología. Queremos restaurar un balance. Estamos de acuerdo con modelar pero no para que algún diagrama quede archivado en un polvoriento repositorio corporativo. Estamos de acuerdo con documentar pero no con cientos de páginas de volúmenes desactualizados y rara vez utilizados. Planificamos pero reconocemos los límites de la planificación en un ambiente turbulento."
>
> 2001 http://agilemanifesto.org/history.html

Principios del modelo ágil de desarrollo

El mismo manifiesto establece doce principios de cualquier modelo de desarrollo ágil de software:

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor [para el cliente].
2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los responsables de negocio [clientes, usuarios] y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7. El software funcionando es la medida principal de progreso [del proyecto].
8. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

http://agilemanifesto.org/iso/es/principles.html Consultado 4 de mayo 2013

El modelo ágil en XP

XP (*eXtreme Programming*) incorpora uno de los procesos más simples y radicales del modelo ágil de desarrollo de software. XP integra un pequeño número de prácticas, entre las que destacan:

1. Programación por pares;
2. Programación dirigida por casos de prueba con refactorización continua;
3. Incorporación permanente de un representante del cliente, especialista en el dominio de la aplicación quien se encarga de escribir las *historias de uso* y las pruebas de aceptación;
4. Entregas incrementales frecuentes, cada 1-4 semanas;
5. Integración continua de software;
6. Planificación incremental
7. Todos los miembros del equipo son dueños y responsables, en común, del código y su desarrollo; cualquiera de ellos puede modificar cualquier cosa.
8. Se evita caer en el hábito del sobretiempo.

XP se fundamenta en cuatro *valores:*

- Comunicación.

  XP incluye prácticas que incentivan la comunicación continua y abierta, tanto entre los desarrolladores como entre los desarrolladores y el cliente tales como la elaboración de *historias de uso,* la programación en pareja y la presencia permanente de un representante del cliente durante el proceso de desarrollo.

- Simplicidad

  El código debe hacerse lo más simple posible; en cuanto empieza a introducirse complejidad innecesaria, los desarrolladores deben refactorizar para simplificarlo. Por eso sólo se documenta lo que se considera indispensable, por eso se tiene cuidado de no proliferar los roles oficiales de los miembros del equipo de desarrollo.

- Retroalimentación

  En un ambiente XP, la retroalimentación está a la orden del día. Cualquiera puede opinar sobre cosa. Se retroalimenta lo más rápidamente posible. Por eso, se elaboran casos de prueba antes de programar (de esta elaboración se tiene retroalimentación sobre las historia de uso, y al ejecutar las pruebas obtenemos retroalimentación sobre el software), por eso se entrega software que funciona cada semana o cada dos semanas, por eso el equipo de desarrollo se reúne brevemente todos los días antes de empezar a trabajar, por eso se programa en parejas, por eso se avisa al grupo si empieza a retrasarse la implntación de alguna de las historias de uso, por eso se arma el sistema completo (*build*) al una o más veces al día , por eso el representante del cliente está presente durante el desarrollo, para ser consultado y consultar en cualquier momento.

- Valentía

  Se requiere valentía para trabajar en un proyecto XP: trabajas a la vista de otro, pruebas la calidad de tu código en forma continúa, recibes retroalimentación de todos, refactorizas todo el tiempo y tienes que estar dispuesto a rehacer tu trabajo si encuentras que el diseño o el código no da la talla o que aparece un nuevo requerimiento que echa por tierra todas las suposiciones que has incorporado al código.

Veamos como XP toma en cuenta cada uno de los doce principios de los modelos ágiles:

Principios del modelo ágil	**XP**
Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.	El cliente decide la prioridad en la implementación de las historias de uso. Se le entrega una nueva versión del software desarrollado al cliente cada una a cuatro semanas. Toda decisión de negocio es tomada por el cliente.
Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.	El cliente puede introducir un nuevo requisito en todo momento. Los desarrolladores estiman el tiempo que les llevaría implementarlo y el cliente decide si lo acepta.

Se entrega software que funciona frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.	Prevé que las entregas se hacen cada 1-4 semanas.
Los responsables de negocio y los desarrolladores trabajan juntos de forma cotidiana durante todo el proyecto.	Un representante del cliente debe estar asignado permanentemente al proyecto para interactuar con los desarrolladores y forma parte del equipo de desarrollo.
Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.	¡Crucial! El programador que se hace responsable de implementar una historia de usuario o una porción de historia de usuario es quien estima el tiempo que requiere para hacerlo. Se le proporciona retroalimentación sobre la precisión de sus estimaciones para que las pueda ajustar. El foco está en la enseñanza de estrategias para mejorar la efectividad de los miembros del equipo, no en la imposición de normas o lineamientos. Se debe motivar continuamente al equipo para que "juegue a ganar". Se incentiva la adaptación a condiciones especiales.
El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.	La programación en pareja permite y requiere de conversación cara a cara. Se tiene una reunión presencial de todos los miembros al menos una vez al día antes de comenzar a trabajar. Se presta particular atención a la configuración física de la oficina[2]
El software funcionando es la medida principal de progreso	El número de historias de uso que ya ejecuta el software es la medida.
Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios deben ser capaces de mantener un ritmo constante de forma indefinida.	Se trabajan un máximo de cuarenta horas semanales. Si, por circunstancias excepcionales, un miembro del equipo trabaja más de cuarenta horas en una semana, de ninguna manera puede trabajar más de cuarenta la siguiente. La programación en pareja, el cambio rutinario de pareja, la simplificidad del código refactorizado incrementan la posibilidades de continuar el desarrollo en forma sostenible.

2 Véase el capítulo 13 (*Facilities Management*) del libro de Kent Beck para más detalles.

La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.	La programación dirigida por casos de prueba y la refactorización continua contribuyen fuertemente a esto.
La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.	La filosofía XP de desarrollar hoy lo que hace falta hoy, la retroalimentación de un entrenador de XP que está pendiente que se cumplan los lineamientos XP y la refactorización continua implementan el principio. El foco de XP siempre está sobre cambios incrementales.
Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.	XP no se basa en roles como jefe de analistas o diseñador de arquitectura sino que busca empoderar a cada miembro del equipo -de hecho cualquier miembro puede introducir una mejora al código (siempre y cuando no lo "rompa" -es decir pase todos sus casos de prueba).
A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.	Cada vez que comienza una nueva iteración, a lo sumo cada cuatro semanas. Si el proyecto se abandona o cancela se incentiva un análisis final con todos los participantes sobre el desempeño del proyecto y los aprendizajes logrados por el equipo.

Ventajas, desventajas y riesgos resaltantes de XP

XP estimula la formación de equipos altamente motivados, compenetrados y flexibles. Pueden ser muy productivos en términos de poder rápidamente producir software que funciona y que le sirve al cliente. El cliente es quien decide qué es lo que quiere que se desarrolle en la próxima iteración y recibe retroalimentación temprana sobre la dirección en que se dirige el software y cómo luce -lo que le permite precisar sus requerimientos o cambiarlos en cuánto se dé cuenta que lo que pidió no es lo que necesita. El software producido es mínimo, en el sentido que no está lleno de cosas que no hacen faltan ahora, lo que facilita entenderlo. Un conjunto de casos de prueba unitarios siempre acompañan el código entregado y forman parte importante de pruebas de regresión. El equipo es robusto en el sentido que no hay ninguna parte del código que sea propiedad exclusiva de un sólo miembro del equipo.

Las desventajas de XP son en buena medida, las representativas de muchos modelos ágiles de desarrollo. No todo el mundo se acostumbra a trabajar en pareja y pueden darse conflictos importantes que afectan la motivación del equipo. Es fácil descartar algunas de las técnicas o prácticas de XP en nombre de estarlo adaptando a las condiciones locales, pero las técnicas están tan acopladas que el riesgo es alto de que se cause, sin querer, un efecto adverso importante. Para muchos programadores trabajar en pareja resulta altamente estresante, la presencia permanente de un representante del cliente puede no ser factible -y esa presencia es una pieza clave en XP. A la hora de las chiquitas, la documentación puede ser inadecuada y a medida que crece el software, la arquitectura plasmada en el software puede resultar innecesariamente frágil.

Como estilo de desarrollo, XP es un estilo arriesgado y tanto el grupo desarrollador como su gerente necesitan hacerle una cuidadosa gestión de riesgos. Algunos de los riesgos más destacados son:

- El representante del cliente no resulta representativo del cliente o no tiene la autoridad para lograr que el proyecto sea aceptado.
- El representante del cliente no tiene tiempo para atender al grupo de desarrolladores pues tiene que "apagar fuegos" propios de su cargo;
- Hay programadores que no quieren trabajar en pareja o que lo hacen a disgusto. Si no es una cuestión que puede resolverse mediante entrenamiento o educación, Beck recomienda sacar al programador del grupo lo más rápidamente posible.
- A los nuevos miembros del grupo les puede costar incorporarse a un grupo ya consolidado.
- Los casos de prueba pueden no estar completos (los programadores no se han dado cuenta de una partición de los datos que debe procesarse de manera diferente al resto);
- La refactorización se puede estar sacrificando en aras de cumplir con fecha de entrega o de avanzar; sobre todo las refactorizaciones más grandes y las que obligan a abandonar la arquitectura vigente.
- Los momentos en que se "rompe" la arquitectura pueden ser muy desmotivantes.
- Puede haber tendencia a arriesgarse a desarrollar con herramientas o en plataformas que se desconocen o que se queden cortos -XP puede estimular una actitud más propensa al riesgo que lo recomendable, lo que conduce a correr riesgos innecesarios.
- El diseño puede quedar enterrado en el código y en la cabeza de los participantes.

Iteraciones, fases y dinámica de desarrollo de XP

Idealmente el proyecto XP pasa por tres fases:

- Una breve fase de desarrollo inicial;
- La fase de producción
- La fase de mantenimiento
- La fase de retiro

En la fase de desarrollo inicial podemos distinguir etapas de exploración, planificación e iteraciones a la primera liberación:

- En la etapa de exploración los desarrolladores forman equipo, se familiarizan o se ponen al día con las herramientas que van a usar y exploran posibilidades para la arquitectura del sistema. El representante del cliente se sienta con los desarrolladores y escribe un primer grupo de *historias de uso.* Una vez que hay suficientes historias como para empezar y que el equipo está preparado en las herramientas, un período que puede ser de un par de semanas o un par de meses, se arranca el proyecto.
- En la etapa de planificación, el cliente selecciona el conjunto de historias de uso que quiere ver en la primera liberación (*release*) del sistema, la cuál debe llevarse entre dos y seis meses. Los programadores estiman el tiempo que creen que les llevará implementar tales historias. Hay interacción entre las dos partes hasta llegar a un acuerdo -en XP la tendencia siempre es hacia respectar la estimación del programador -por ende el cliente puede terminar por aceptar fechas

diferentes a las que él tenía en mente o redefine el alcance de lo que se incluirá en la primera liberación. Esta etapa dura 1 a 2 días.

- En la etapa de elaboración de la primera liberación, se planifican *iteraciones* que duran entre 1 a cuatro semanas. En la primera iteración se busca plasmar los grandes lineamientos de la arquitectura y se escogen historias de uso que obliguen a "ver" el sistema. Después de la primera iteración, es el cliente que decide qué historias son las más prioritarias. Recuerde que en cualquier momento el cliente puede cambiarle la prioridad a las historias o cambar los requerimientos.

En la fase de desarrollo para entrar en producción típicamente se pasa a iteraciones *semanales* y se presta particular cuidado a las pruebas de aceptación y a entrenar los usuarios para que puedan usar el software cuando oficialmente entre en producción.
La fase de mantenimiento es la fase "usual" en que se encuentra un proyecto XP. Simultáneamente hay que seguirle agregando funcionalidad al software, mantenerlo operando y tratar con la (inevitable) rotación de personal. Se necesita crear un "escritorio de ayuda" y desarrollar un sistema de turnos para que los desarrolladores atiendan los usuarios cuando estos necesiten consultar dudas y sugerir mejoras al sistema.
En la fase de retiro se agotan las historias de uso por lo que es tiempo de suspender el desarrollo. Se aprovecha este momento para documentar el sistema lo suficientemente bien como para facilitar retomar su desarrollo en uno o dos años cuando vuelvan a surgir requerimientos de cambio. También puede darse el caso que se decida retirar al software -o no da la talla o el cliente necesita otra cosa. En todo caso es conveniente hacer un análisis final del sistema para dejar constancia de sus aciertos y de sus desaciertos de modo de aprender de él lo suficiente para mejorar el desarrollo de futuros proyectos.

El modelo ágil en Scrum

[Pendiente por desarrollar]

Limitaciones del modelo ágil de desarrollo

Los modelos ágiles de desarrollo de software no son para todo el mundo, ni son para todos los proyectos. Una de las más difíciles lecciones que se ha aprendido en la última década es que el proceso de desarrollo necesita ajustarse a las características de los proyectos de desarrollo y a las personas con que se cuenta.

Somerville resume algunas de las limitaciones con este modelo:

1. El modelo ágil de desarrollo requiere que sus miembros se involucren intensamente en el proyecto y con los demás miembros del equipo -no todo el mundo se presta a ello. Por ejemplo la práctica de programación en pareja requiere de programadores técnicamente talentosos, con habilidades comunicativas y personalidades poco abrasivas y con autoestima.
2. Es fundamental que un representante experto del cliente esté comprometido con el proyecto y se involucre a dedicación casi exclusiva con él, formando parte del proyecto de desarrollo y colaborando con los demás miembros. Esto no siempre es posible -a veces la empresa-cliente no puede permitirle a la persona clave que deje sus responsabilidades cotidianas para formar parte del desarrollo. Este representante debe tener el empoderamiento y la autoridad necesaria para que sus decisiones conduzcan a un software que será aceptado, y no resistido, por sus pares y los usuarios del software.

3. Pueden haber serios desacuerdos sobre las prioridades a otorgarles a los cambios solicitados, sobre todo cuando se presentan divergencias de opinión entre los propios clientes y usuarios.
4. La presión de entregar puede comprometer la etapa de refactorización.
5. La empresa cliente necesita confiar en la metodología y en los desarrolladores y puede considerar que el contrato típico para un desarrollo ágil es demasiado etéreo para poder autorizarlo.
6. Hay dominios en el que las exigencias de seguridad y estabilidad del software y su co-dependencia con hardware de control, obliga a que se analicen cuidadosamente la completitud, consistencia y dependencia entre los requerimientos, antes de comenzar el desarrollo. Estos tipos de dominios no se prestan a modelos ágiles de desarrollo.
7. Para el 2002, Boehm reportó que el 80% del retrabajo realizado en software de gran envergadura se originaba en situaciones donde la arquitectura del software se había "roto" debido a que no lograba satisfacer requerimientos de desempeño, tolerancia a fallas o seguridad. Esto sugiere que hay circunstancias en que dejar el diseño de mañana para mañana no es la mejor estrategia, sobre todo cuando se trata de diseño arquitectónico.
8. Boehm también menciona en el 2001, en los tiempos iniciales del modelo ágil, que el modelo parece dar sus mejores resultados con hasta 15-20 desarrolladores, aunque también reportó algunos proyectos ágiles exitosos que involucraron hasta 250 personas.

Lecturas adicionales

- Ian Sommerville: *Software Engineering.* 8° edición. Addison-Wesley, 2007. El capítulo 17 (*Rapid Software Development*) incluye dos secciones (*17.1 Agile Methods* y *17.2 Extreme Programming*) que constituyen una excelente introducción al tema.
- Roger S. Pressman: *Software Engineering: A Practitioner´s Approach.* 7° Edición, McGraw-Hill, 2010. El capítulo 3 (*Agile Development*) constituye una cobertura comparable a la del texto de Sommerville con algunas interesantes variantes.
- Scott Duncan: *Engineering Practice and Bridge Design.* Agile Software Qualities, 17 de agosto 2009. http://agilesoftwarequalities.blogspot.com/2009/08/engineering-practice-and-bridge-design.html Consultado el 4 de mayo 2013. Una interesante revisión de la metáfora de la ingeniería de puentes a la luz del estado de los modelos ágiles de desarrollo de software.
- Kent Beck: *eXtreme Programming explained: Embrace Change.* Addison-Wesley, 2000. Sigue siendo a mi juicio una de las más claras y motivantes introducciones a un modelo ágil de desarrollo de software. [Algunos reseñadores de *Goodreads* comentan que la segunda edición introduce ajustes importantes.]
- Barry Boehm: *Get Ready for Agile Methods, with Care.* IEEE Computer, Enero 2002. http://sunset.usc.edu/events/2002/arr/Get%20Ready%20for%20Agiel%20Methods,%20with%20Care.pdf Consultado 4 de mayo 2013. Una temprana pero balanceada evaluación de los modelos ágiles.
- http://www.agiledata.org/essays/agileDataModeling.html Un excelelente artículo sobre cómo pueden aplicarse los modelos ágiles al desarrollo de bases de datos.
- *Manifesto for Software Craftmanship*, 2009. http://manifesto.softwarecraftsmanship.org/

Consultado 4 de mayo 2013. Una interesante complementación del Manifiesto Ágil a ocho años de la aparición del original.