Los peligros de los patrones de diseño

Fecha de creación: 17.12.2002

Revisión 1.0 (17.12.2002)

Martín Pérez Mariñán (martin AT javahispano DOT org)

http://www.javaHispano.org

Los peligros de los patrones de diseño

Actualmente, los patrones de diseño[1] son sin duda alguna la herramienta más importante de la que disponemos los ingenieros, arquitectos, analistas, desarrolladores, etc., para la creación de sistemas robustos, escalables, fácilmente adaptables y con grandes cotas de reutilización. Se han escrito miles y miles de líneas sobre las ventajas de los patrones de diseño, pero sin embargo, pocos han sido los análisis de los problemas asociados a su uso. En este pequeño artículo intentaré mostrar estos problemas, y que de este modo este artículo sirva para no cometer alguno de los errores más comunes que se producen al aplicar patrones de diseño.

Una pequeña y exagerada historia ficticia

Esta pequeña historia que voy a contar, aunque algo exagerada, servirá como introducción a los problemas que debemos afrontar los desarrolladores de software a la hora de utilizar patrones de diseño. En ningún momento trata de reflejar una historia real, aunque podría darse en cualquier momento y probablemente hayamos visto reflejada la idea subyacente bajo dicha historia en muchas situaciones diferentes.

Dos ingenieros que acaban de terminar la carrera coinciden en una entrevista de trabajo de una importante compañía. Ambos pasaron las primeras pruebas y se encuentran en la última fase en la cuál uno de los responsables del equipo de desarrollo les ha preparado un pequeño examen en el que comprobará cuáles son sus conocimientos sobre temas como ingeniería de software, gestión y desarrollo de proyectos, etc.

El examen consta de diez preguntas y ambos comienzan a realizarlo con suma tranquilidad. Las primeras preguntas tocan temas que conocen de sobra ya que ambos tienen una gran preparación teórica. Nuestros amigos responden con seguridad y firmeza a las primeras preguntas sobre UML, patrones y métricas. Todo va bien hasta que ambos llegan a la pregunta número cinco:

Diseña un programa que imprima la frase "Hola Mundo" lo mejor que te sea posible

Desde luego no se esperaban una pregunta como esta. Qué soberana tontería piensan ambos. En realidad tienen toda la razón, pero sin embargo, no saben que responder.

El primero de los dos ingenieros toma la decisión de resolverlo de la manera más simple posible y decide responder a la pregunta con varias líneas en las que codifica la solución en su lenguaje de programación favorito.

El segundo ingeniero sin embargo decide tomar otro camino. *Esto tiene truco, como estoy en una gran empresa quieren ver si somos capaces de afrontar de una manera elegante los retos que hemos de afrontar para que de este modo nuestros sistemas siempre estén preparados para todo y tengan un alto grado de flexibilidad*, piensa para si mismo al tiempo que comienza a realizar un diseño del problema basado completamente en patrones y prácticas deseadas.

Así, nuestro amigo ha decidido crear una separación MVC de modo que el programa lee la frase "Hola Mundo" desde un fichero XML y la trata, consiguiendo de este modo una

perfecta separación entre la lógica de negocio y la presentación. Además, ha decidido crear una cadena de responsabilidad de modo que la frase pueda ser tratada por diferentes objetos según queramos que salga por pantalla, se guarde en disco o se envíe por una red, todos esos objetos provienen de diferentes familias que obtiene con un patrón factoría abstracta.

Cuando ha finalizado el diseño está muy orgulloso de él, se van a quedar impresionados, piensa. Sin embargo, en el momento de llegar a la sexta pregunta la persona encargada de llevar la entrevista dice que se ha acabado el tiempo.

> *[Ingeniero 2] : Pero como puede ser, no nos había avisado que había tiempo límite*

> *[Examinador]: Por supuesto, en esta prueba intentamos comprobar sus habilidades a la hora de diseñar proyectos. Nuestras empresas siempre nos piden las cosas mucho antes de que estén terminadas. Una de las cualidades de las personas que buscamos es saber aprovechar al máximo los márgenes de tiempo de que disponen, incluso cuando estos no se conocen de antemano.*

> *[Ingeniero 2]: Pero es injusto. He estado diseñando un sistema muy avanzado para que sean capaces de apreciar todo lo que sé.*

El examinador observa un par de segundos el examen de nuestro amigo y responde:

> *[Examinador]: Desde luego que es un diseño fantástico, digno de un trabajo de ingeniería y del mejor creador de proyectos del que disponemos. Sin embargo no es arte lo que estamos buscando sino pragmatismo.*

Que paradoja, después de haber creado una pequeña e ingeniosa maravilla, el segundo ingeniero no obtendrá el trabajo. El primer ingeniero tampoco tuvo tiempo de contestar todas las preguntas. Realmente sólo fue capaz de contestar a ocho de las mismas, ya que también estuvo bastante tiempo pensando la solución de la quinta pregunta, sin embargo supo aprovechar mucho mejor el tiempo disponible y por eso, finalmente, fue el que consiguió entrar a trabajar en la prestigiosa empresa.

Moraleja : Hay que saber resistirse a la euforia

Si hay una sensación que describa lo que puede sentir un desarrollador después de conocer los patrones de diseño esa podría ser la euforia. Esta euforia viene producida por haber encontrado un mecanismo de ingeniería que convierte lo que era una labor artesana y tediosa en un proceso sólido y basado en estándares, gracias a la ayuda de lenguajes de modelado y metodologías.

Después de que un recién iniciado a los patrones de diseño haya apreciado las maravillas de este mecanismo, probablemente el paso siguiente que tomará será emprender el rediseño de algunos proyectos aún vigentes y que estuviese realizando aplicándole todas las maravillosas técnicas que ha aprendido para que así estos proyectos se aprovechen de todos los beneficios inherentes al uso de patrones de diseño.

Lamentablemente, si este es el camino decidido, una vez haya finalizado el proceso de

refactorización[2][3], comenzarán los problemas. Seguramente lo primero que se encontrará es que ha utilizado algunos patrones incorrectamente, tal vez porque no lo haya entendido o porque lo ha aplicado sin pensar demasiado, y se verá obligado a reorganizar el proyecto, readaptar el patrón de diseño, o aún peor, mantendrá ese patrón híbrido que no se corresponde con ningún estándar.

Posteriormente, después de adquirir más experiencia o consultar bibliografía especializada al problema en cuestión, seguramente se encuentre con que en el lugar donde había aplicado el patrón X nos había ido mucho mejor con el patrón Y, esto generará un nuevo proceso de refactorización, y que al contrario del anterior puede que lleve todavía mucho más tiempo, ya que los patrones híbridos pueden haber creado dependencias indeseables. Poco a poco, la cohesión entre los elementos irá aumentando, comenzarán a aparecer malos hábitos, y lo que en un inicio habíamos ideado como una manera de mejorar el diseño de nuestro código degenerará en justamente lo contrario, es decir, en código incompresible e inmanejable.

¿Qué ha pasado? ¿No eran los patrones de diseño la navaja suiza que haría nuestro trabajo mucho más provechoso sobre la base de conceptos como la reutilización? La respuesta a esta pregunta es muy simple: "Sí, pero fundamentados en un análisis meticuloso del dominio del problema, en un aprendizaje continuo y en la experiencia".

Veamos más detenidamente cada una de las partes de esta frase:

» **Análisis:** Diseñar una arquitectura basada en patrones de diseño es una tarea compleja, que reporta muchos beneficios pero prácticamente imposible sin un estudio previo del dominio del problema que estamos tratando[4]. El uso de lenguajes de modelado como UML nos ayuda a ver mejor las propiedades del sistema que estamos diseñando y harán aparecer relaciones entre los componentes que a primera vista eran difusas o que simplemente no habíamos visto.

El mero hecho de utilizar patrones de diseño "porque sí" y comenzar a realizar su implementación sin tener en cuenta el dominio de nuestro modelo, es un error gravísimo y degenera en la práctica de ir esparciendo patrones por las diferentes partes de un proyecto sin ningún criterio general. Estos patrones no representarán ningún concepto de nuestro dominio, probablemente no reflejarán ningún significado a alguien que no esté familiarizado con nuestro proyecto, y dificultarán notablemente la extensibilidad de nuestro proyecto obligando a una refactorización continua.

Antes de abordar un proyecto con patrones hemos de analizar minuciosamente qué patrones nos pueden ser útiles, cuáles son las relaciones entre los diferentes componentes de nuestro sistema, cómo podemos relacionar los patrones entre sí de modo que formen una estructura sólida, cuáles son los patrones que refleja nuestro dominio, etc. Esta tarea obviamente es mucho más compleja que el mero hecho de ponerse a codificar patrones "porque sí".

» **Aprendizaje continuo:** A menudo uno piensa que como conoce los patrones GOF[1], ya tiene en su mano un artificio que le permitirá superar cualquier problema que se le presente, sin darse cuenta que tiene en sus mano una espada de Damócles que puede volverse fácilmente en su contra.

Una gran equivocación. Los patrones, por definición, aparecen en cualquier

situación y no están sujetos a ataduras que nos digan "esto es lo que hay y no existe nada más". Existen cientos y cientos de patrones que han ido apareciendo con el tiempo y que se han convertido en soluciones estándar para diferentes problemas, y por supuesto no limitados al ámbito informático sino que podemos encontrar patrones de arquitectura, de economía, etc.

Por poner un ejemplo, si nos estamos enfrentando a un problema dentro del ámbito de la mensajería, lo más adecuado es comencemos por estudiar patrones comunes en sistemas de mensajería en lugar de intentar empezar desde cero aplicando patrones como un Observador. Así descubriremos nuevos patrones que engrosarán nuestra caja de herramientas, patrones que por otra parte se han mostrado efectivos ante el problema en concreto que estamos tratando.

» **Experiencia:** La clave de todo. Puede que hayamos realizado un análisis fantástico y que hayamos estudiado los patrones más avanzados en el ámbito que estamos afrontando, sin embargo, sin la experiencia adecuada nuestro proyecto fracasará.

Sea como sea, la experiencia se adquiere con el tiempo y a base de tropiezos. Seguramente nuestros primeros proyectos no queden como los habíamos ideado en su concepción, y con el tiempo, nos demos cuenta de que podríamos haber utilizado otros métodos para abordar de un modo más eficiente los problemas con los que nos habíamos enfrentado. La experiencia es un grado muy importante y que con el tiempo sabremos aprovechar para llevar nuestros proyectos adelante.

En los siguientes puntos vamos a analizar algunos de los problemas más comunes derivados del uso de patrones de diseño sin tener en cuenta las pautas de utilización descritas anteriormente. Además, expondremos soluciones prácticas para cada uno de estos problemas.

El problema: Demasiado ingenio

La sobre-ingeniería, del término inglés overengineering, es seguramente el problema más importante derivado del mal uso de los patrones de diseño. Al igual que en el ejemplo anterior nuestro amigo quiso crear una arquitectura compleja para resolver un problema muy simple, es probable que ante un problema que se nos presente nos veamos tentados a crear una arquitectura compleja, sofisticada, y por supuesto, muy elegante, para un problema que podría ser resuelto en cuestión de horas.

La complejidad de los diseños, a menudo hace que los desarrolladores se vean sobrecargados, sobre todo cuando en lo que pensaban que podía ser un diseño flexible y elegante comiencen a aparecer casos especiales que obligan a un proceso de refactorización continuo. Normalmente este proceso de refactorización degenerará en una distorsión de nuestro modelo e irá añadiendo capas de dependencias entre los diferentes componentes. Además, esto se ve agravado por la complejidad del diseño realizado que hace que dicha refactorización no sea una tarea simple y que tiene como consecuencia graves retrasos en los proyectos.

La solución: Agudizar el ingenio para crear diseños menos

complejos

¿ Quién tiene más ingenio, un desarrollador que soluciona un problema de una manera simple y sencilla o un desarrollador que soluciona el mismo problema dos meses más tarde pero con una arquitectura tan excepcional como innecesaria ?

Es mucho más ingenioso, y a menudo más difícil, crear una solución simple que una extremadamente compleja para un problema determinado. Los proyectos han de ser lo más sencillos que nos sea posible. Hemos de evitar complejidades innecesarias, porque éstas acabarán generándonos grandes costes en cuanto a refactorización y mantenimiento.

Con sistemas simples y que se ajusten a los problemas que queremos resolver, la posibilidad de que aparezcan casos no contemplados disminuye de manera exponencial. Es muy fácil que se nos escapen pequeños problemas en partes no necesarias y que en un futuro nos pueden crear grandes dificultades, por eso lo mejor es mantenerse siempre fiel a las necesidades fundamentales del proyecto.

Esta solución es una de las premisas que tratan de imponer las metodologías ágiles[6], que predican la creación de modelos sencillos que paulatinamente se irán convirtiendo en modelos más complejos siguiendo un proceso de refactorización iterativo que mantenga intacta la simplicidad global del sistema.

El problema: Duración de los proyectos

Una consecuencia de la sobre-ingeniería que tratamos en el punto anterior es el retraso en los proyectos. La creación de sistemas complejos tiene un coste temporal asociado que a menudo conlleva retrasos en la finalización de los proyectos. Los patrones de diseño añaden inherentemente un coste de complejidad que puede que en algunos proyectos puntuales no sea necesario asumir.

Lamentablemente, en el mundo del desarrollo informático los profesionales estamos sometidos a un continuo estrés temporal. Conforme pasa el tiempo, los proyectos requieren más y más funcionalidades y a poder ser que estén disponibles en el menor tiempo posible[5]. Un mal uso de los patrones de diseño y en especial su uso cuando no es necesario pueden alargar considerablemente los plazos de desarrollo y hacer fracasar un proyecto.

La solución: Desarrollo ágil, ajustarse a los plazos temporales

Lamentablemente, es una realidad que la mayor parte de los proyectos no llegan, o lo hacen con retraso, a su meta final. Los nuevos modelos ágiles de desarrollo, están intentando reducir la aparición de estos problemas. Controlando la evolución de nuestros desarrollos, de modo que vayamos lanzando poco a poco pequeñas versiones simples que crecen en complejidad de manera controlada, minimizamos la posibilidad de aparición de retrasos temporales.

La clave está en la simplicidad. Hemos de empezar con versiones simples y que se ajusten lo más posible al problema que estamos tratando. Esas versiones serán refactorizadas constantemente en un proceso iterativo en base a pequeños pasos y siempre manteniéndonos fieles al problema que estamos tratando. Las iteraciones han de ser frecuentes, de este modo, se controla mejor la evolución del ciclo de desarrollo y se

pueden detectar más fácilmente cuales son los puntos que pueden retrasar el proyecto.

El problema: Escasa mantenibilidad

Si nuestro equipo de desarrolladores no tiene la suficiente experiencia, crear arquitecturas complejas puede ser contraproducente. A la hora de mantener un sistema, si la persona que va a realizar la tarea no entiende dicho sistema debido a su complejidad, a su extensión o a que usa mecanismos demasiado sofisticados que simplemente no conoce, la tarea de mantenimiento puede alargarse mucho más de lo que fuera deseable.

De nada vale utilizar patrones de diseño si nadie los comprende en un equipo de desarrolladores ya que se crearán sistemas inmanejables y se crearán dependencias en los proyectos. Si un jefe de proyecto comienza a crear una arquitectura sofisticada y a continuación se la presenta a su equipo de desarrolladores para que comiencen su creación y éstos no entienden lo que están leyendo pueden pasar varias cosas:

» Que reconozcan que no tienen esos conocimientos: Esto impone unos retrasos considerables en el proyecto ya que será necesaria una formación previa de los desarrolladores. Además, debido a su poca experiencia se producirán el resto de problemas que estamos tratando.

» · Que no digan nada: Este caso es todavía peor, ya que los desarrolladores comenzarán a realizar el desarrollo "buscándose la vida". Las consecuencias son terribles: se pierde totalmente la relación entre el modelo y la implementación, uso de patrones híbridos, escasa reutilización, etc.

La solución: Control, cooperación y formación continua

Los diseños "artísticos" son siempre un problema y deben ser evitados a toda costa. Un proyecto muy ambicioso puede fracasar simplemente porque el desarrollador líder haya creado una arquitectura compleja y sofisticada y después haya decidido abandonar la empresa, dejando en manos desarrolladores menos experimentados un diseño excepcional pero inmanejable. Es necesario una gran dosis de autocontrol por parte de los desarrolladores más experimentados para no sobrecargar a sus compañeros.

El desarrollo es una labor de equipo. De nada sirven las grandes individualidades si el equipo es incapaz de aprovecharlas. Como labor de equipo, éste ha de ser muy consciente de sus limitaciones y tratar de ceñirse a lo que está capacitado para realizar conjuntamente. Estándares, convenciones, lenguajes comunes, todos son artificios que pueden incrementar notablemente la cooperatividad entre los miembros del equipo, fomentando la compartición de ideas y de conocimiento e implícitamente incrementando la mantenibilidad de los proyectos.

Aún así, los patrones de diseño son una herramienta imprescindible por todas las ventajas que nos aportan, demasiadas como para renunciar a ellos. Si nuestro equipo no domina los patrones de diseño hemos de tratar de ir incorporando poco a poco su conocimiento a base de formación continuada. Primero intentaremos implantar patrones simples y poco a poco iremos formando a nuestro equipo para que pueda asumir patrones más complejos. Todo esto puede realizarse de manera solapada a las diversas iteraciones de nuestro plan de desarrollo.

El problema: Perdida de claridad

A menudo la extensión y complejidad de un proyecto son cualidades inversamente proporcionales a su claridad. Diseños y modelos complejos normalmente tienen muchas carencias en cuanto a comunicación y explicación. Estos diseños "artísticos" normalmente no reflejan de una manera clara y simple el modelo del dominio que se está tratando.

Asimismo, y como también ya hemos recalcado, el uso de patrones sin razón de ser en un modelo determinado no hacen más que ofuscarlo. El modelo pasa a incluir conceptos que no tienen ninguna relación con el dominio que se está tratando y está perdiendo su relación con éste.

Las consecuencias de todo esto se resumen en : perdida del significado del modelo. Personas acostumbradas al dominio en el que nos encontramos, típicamente conocidos como expertos del dominio, serán capaces de entender lo que hemos modelado. A raíz de esa falta de claridad poco a poco el modelo se irá complicando cada vez más y reflejará vanamente el dominio del problema que intenta solucionar.

La solución: Adaptarse al modelo del dominio

Un diseño, un modelo conceptual, y en general un proyecto ha de ser claro. Crear una tela de araña de patrones de diseño donde se podría haber utilizado una estructura mucho más simple no hace más que entorpecer la comprensión de la arquitectura del sistema. Diseños claros son mucho más sencillos de comunicar y expresan por si mismos las características del sistema que modelan.

Los modelos han de ser lo suficientemente claros como para que personas no acostumbradas al mundo de la programación sean capaces de entenderlos. Se ha de facilitar la comprensión de dicho modelo por parte de todos los miembros involucrados en el desarrollo, especialmente de aquellos que conocen el dominio pero no necesariamente han de tener conocimientos técnicos. Para ello los modelos han de reflejar claramente los conceptos del dominio que estamos tratando y ceñirse al mismo.

Problema: Poca experiencia

La experiencia es un grado. Sólo adquiriremos fluidez en el desarrollo basado en patrones cuando los hayamos aplicado en multitud de ocasiones. Existen gran cantidad de pequeños problemas que tienen como raíz la falta de experiencia. Los iremos viendo poco a poco son sus soluciones:

- **Pequeño problema: No conocer los patrones adecuados**

  Como ya vimos antes, probablemente para un problema de mensajería convenga que apliquemos patrones de mensajería que ya se han mostrado exitosos en dicho dominio. Si no conocemos dichos patrones probablemente estemos reinventando la rueda o se nos escapen cosas que otras personas ya han analizado previamente y para las que ya se han encontrado soluciones estándar.

  El desconocimiento de los patrones adecuados por lo tanto nos puede provocar un alargamiento en nuestros proyectos ya que nos veremos obligados a buscar soluciones para problemas que ya han sido tratado, a corregir dichas

soluciones cuando aparezcan casos que no habíamos considerado, etc.

La solución: Buscar patrones para el dominio tratado

En la actualidad son pocos los dominios que no dispongan de una serie de patrones y estándares definidos para el desarrollo. Patrones de mensajería, patrones para sistemas distribuido, patrones para sistemas empresariales, patrones para sistemas móviles o patrones para servicios web, son sólo algunos ejemplos de catálogos de patrones que extienden a los patrones GOF.

Antes de comenzar nuestro diseño hemos de realizar un estudio meticuloso del dominio en el que nos encontramos y explorarlo en busca de patrones que hayan sido utilizados previamente con éxito. Estos patrones nos ayudarán a que nuestro proyecto evolucione mucho más rápidamente.

» **Pequeño problema: Usar los patrones incorrectos**

A menudo la falta de experiencia puede llevarnos a utilizar un patrón en algún lugar donde habría sido mejor utilizar otro patrón. Ejemplos típicos pueden ser utilizar un patrón Builder en lugar de un patrón Abstract Factory, o utilizar un patrón Chain of Responsability en lugar de aplicar una cadena de Decorator, etc.

El uso de patrones incorrectos puede provocar serias inconsistencias en nuestro modelo y la refactorización del mismo pueden ser bastante compleja, debido principalmente a que pueden haberse introducido diferentes dependencias en cadena.

La solución: Buscar soluciones para problemas similares. Mentoring.

Para no realizar la elección equivocada siempre es conveniente analizar las soluciones que se han tomado previamente a problemas similares al que estamos abordando. Con esta medida evitamos caer en equivocaciones, siempre y cuando nos basemos en sistemas que han resultado exitosos.

Otra medida muy importante es la de dejarnos aconsejar por un mentor adecuado. Si no disponemos del conocimiento suficiente, no es una mala idea exponerle el problema a un experto o a una consultoría especializada en patrones de diseño. Además de obtener una solución avalada por dichos expertos seguramente podamos aprender muchos conocimientos y buenas prácticas de estas personas.

» **Pequeño problema: Usar patrones de manera inadecuada**

Como ya se mencionó anteriormente, la euforia producida al descubrir los patrones de diseño puede hacer que nos pongamos a crear patrones "a lo loco". Estos patrones normalmente no son tal, y simplemente son modelos que reflejan algunas características especificas (probablemente las más sencillas)de algún patrón determinado.

Estos patrones híbridos, fruto de querer implantarlos de una manera rápida y sin un análisis detallado del patrón, provocan graves problemas en el código, ya que hacen que creamos que estamos utilizando patrones de diseño cuando en realidad no lo estamos haciendo. Problemas de mantenibilidad, extensibilidad, flexibilidad son sólo unos pocos de los que se nos pueden presentar.

La solución: Estudio continuado

La solución se reduce simplemente al estudio de los patrones de diseño. Existen libros o sitios en la red donde vienen ampliamente detallados los patrones de diseño más utilizados. Hemos de analizarlos, comprenderlos, estudiarlos y aplicarlos en algún ejemplo simple antes de proceder a implantarlos en nuestro modelo, ya que en caso contrario puede que estemos introduciendo alguna inconsistencia que nos cause problemas más serios posteriormente.

Como se puede deducir de los puntos anteriores las soluciones al problema de la experiencia son muy variadas: Estudio del dominio, análisis de los patrones, formación, consultoría especializada, etc. son factores que pueden ayudarnos a ir aumentando nuestra experiencia y que nos resultarán de un valor incalculable en futuros desarrollos.

Conclusión

Los patrones de diseño son una herramienta de gran valor para el desarrollo de software en general. Presentan soluciones estándares a problemas que se plantean en diferentes entornos. Sin embargo, su uso descuidado puede presentar una serie de problemas que pueden provocar el efecto contrario al prometido por estos artilugios: retrasos en el desarrollo de los proyectos, escasa reutilización, problemas en la mantenibilidad, etc.

Saber ajustarnos al dominio del problema tratado limitando nuestras capacidades "artísticas", mantener nuestros modelos lo más simples posibles, realizar desarrollos en equipo eludiendo patrones complejos e incomprensibles por gran parte de sus miembros, saber ajustarnos al modelo del dominio que estamos tratando y tener una formación continua ya sea a base de aprender nuevos patrones, de consultoría especializada o del estudio constante, son algunas de las medidas que pueden evitar los problemas asociados a los patrones de diseño.

En resumen : hemos de tratar de ser más ingenieros y menos ingeniosos.

Recursos y referencias

[1] Design Patterns, Elements of Reusable Object-Oriented Software, Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Addison Wesley
[2] Refactoring, Martin Fowler,
[3] Refactoring to Patterns, Joshua Kerievsky, *http://www.industriallogic.com*
[4] Domain Driven Design, Tackling Complexity in the Heart of Business Software. Eric Evans, *http://www.domainlanguage.com*
[5] Why Software is so Bad ? (July / August 2002), *http://www.technologyreview.com*
[6] Enlaces sobre metodologías ágiles, *http://www.agileprogramming.com*

Acerca del autor

Martín Pérez Mariñán trabaja en España para INTECNO, empresa del Grupo DINSA, desarrollando aplicaciones para el Hospital Juan Canalejo de A Coruña. Martín tiene ya cinco años de experiencia desarrollando aplicaciones con lenguajes orientados a objetos como Java, es Ingeniero de Sistemas por la universidad de A Coruña además de ser Sun Certified Java Programmer y Sun Certified Java Developer.