Sánchez Ortega, I. (2010): Cartografía extrema. En: Ojeda, J., Pita, M.F. y Vallejo, I. (Eds.), *Tecnologías de la Información Geográfica: La Información Geográfica al servicio de los ciudadanos*. Secretariado de Publicaciones de la Universidad de Sevilla. Sevilla. Pp. 1.379-1.384. ISBN: 978-84-472-1294-1

CARTOGRAFÍA EXTREMA

Iván Sánchez Ortega[1]

(1) OpenStreetMap Foundation. ivan@sanchezortega.es

RESUMEN

Durante la última década, las metodologías ágiles de desarrollo de software, como Scrum y la programación extrema, han cobrado relevancia y están modificado la manera en la que se desarrolla software en las empresas, apartándose radicalmente de las metodologías tradicionales de desarrollo en cascada.

Sorprendentemente, muchas de las técnicas ágiles son aplicables también a la producción, gestión y control de calidad de la información geográfica. El conocimiento de estas técnicas proporciona una visión de cómo puede mejorarse en conjunto el ciclo de producción y mantenimiento de la información geográfica, basándose en los cambios que han sufrido los ciclos de producción y mantenimiento de software.

Palabras Clave: programación extrema, agilismo, OpenStreetMap

ABSTRACT

During the last decade, agile methodologies for software development, as Scrum and extreme programming, have become relevant and are chancing the ways in which software is developed, moving away from traditional waterfall development methodologies.

Surprisingly, many of the agile programming techniques can also be applied to geographical information production, management and quality control. Knowledge of these techniques gives a big picture of how the geographical information production and maintenance cicle can be improved as a whole, based in the changes that software development and maintenance have suffered.

Key Words: extreme programming, agile programming, OpenStreetMap

INTRODUCCIÓN

Tradicionalmente, para el desarrollo de software, se ha utilizado una metodología denominada “en cascada”. Este método consiste en realizar una fase después de otra, y empezar una nueva fase sólo después de que la anterior esté totalmente completa. Aunque la distinción entre fases puede variar dependiendo de la metodología específica, típicamente estas fases son diseño, implementación (o codificación), pruebas (o verificación), y despliegue (o implantación).

De manera análoga, los proyectos de índole cartográfica siguen *a grosso modo* estas mismas fases:

1. Diseño: qué tipo de geometrías se van a utilizar, qué campos tienen, qué formatos va a tener el producto final.
2. Trabajo de campo: recogida *in situ* de los datos necesarios.
3. Gabinete: digitalizar los datos recogidos en campo.
4. Control de calidad: Asegurar diversos parámetros de los datos procesados en gabinete.

5. Despliegue: Generación del producto final, y publicación, entrega, o puesta a disposición del mismo.

Estas metodologías, con fases secuenciales, funcionan bien para las ingenierías "tangibles" (por llamarlas de alguna manera), aquellas que producen objetos que podemos tocar y manipular. Por ejemplo, un ingeniero de caminos primero diseñará un puente; sólo después de tener el diseño completo podrá empezar a construirlo; y sólo una vez construído podrá realizar un control de calidad.

Esto es así porque no se puede realizar un control de calidad sin tener el puente construído: es imposible ver cuánto se deforma el puente si no hay puente. De igual manera, no se puede empezar a construir sin haberlo diseñado completamente, pues no se sabría qué materiales utilizar.

En definitiva: en este tipo de ingenierías, el coste de realizar un cambio en una fase ya terminada (por ejemplo, realizar un cambio de diseño en la fase de construcción) es prohibitivamente alto. Es por ello que una metodología en cascada es apropiada para estos casos.

Sin embargo, el caso de las ingenierías "intangibles" (que producen objetos que no se pueden tocar) es bien distinto. En estos casos, sí que es posible empezar una fase antes de haber terminado las anteriores, y el coste del cambio, en contra de lo que pudiera parecer intuitivo, es menor cuanto menos tiempo se tarda en realizar el cambio en las fases anteriores.

Concretamente, en el ámbito de la ingeniería del software, existen estudios en torno a las denominadas *técnicas ágiles*, y en particular, en torno a la *programación extrema* (que consiste en o bien eliminar un aspecto de las metodologías no ágiles, o bien potenciarlo en grado extremo, de ahí el nombre). Merece una especial atención el libro *Extreme Programming Explained: Embrace Change*, en el cual se basa gran parte de este artículo. Este libro narra cómo, utilizando determinadas técnicas que hacen replantearse el flujo de trabajo de un proyecto de ingeniería de software, se puede conseguir un producto de mejor calidad, manteniendo bajos los costes de realizar cambios.

La cuestión que aquí se plantea debería ser obvia. Si tanto la ingeniería del software, como la cartografía, topografía, geomática y demás ciencias afines producen objetos intangibles (es decir, ficheros de ordenador y bases de datos), ¿Son aplicables las técnicas ágiles y de programación extrema al campo de la cartografía?

Intentaremos resolver esta cuestión viendo cómo algunos proyectos ya hacen lo que podríamos denominar como *"Cartografía Extrema"*.

Sentarse juntos ("Sit together")

La primera técnica de programación exrema no tiene nada que ver con la programación, sino con las personas. Consiste en tener un espacio de trabajo abierto, en el que las personas puedan hablar entre sí sin necesidad de tener que llamar a la puerta de un despacho. Algo tan sencillo como unir a toda la gente en un misma sala para que puedan consultarse dudas mutuamente ha ayudado ya a múltiples proyectos de software.

¿Cómo se puede aplicar este principio a la cartografia?

No hay nada que impìda a un equipo de SIG el sentarse en la misma sala. No es raro encontrarse en situaciones en las que haya que pregunar cómo funciona cirto software SIG, o pedir una segunda opinión sobre un conjuno de datos, o sencillamente hacer que el espacio de trabajo tenga un aspecto más humano.

Un ejemplo de esto lo tenemos en la geolocalización de SMSs durante la crisis del terremoto de Haití en un improvisado *crisis camp*. Los SMSs que se enviaban a la central de emergencias eran traducidos del criollo al inglés, y posteriormente georreferenciados. Buscar constantemente la localización geográfica de puntos de interés o direcciones postales puede ser bastante tedioso, pero si se hace en grupo, todas las personas involucradas tienen una mayor sensación de estar haciendo algo útil, además de poder pasarse de unos a otros la carga de trabajo, dependiendo de si alguna persona se especializaba en algún área de las ciudades o algún tipo de entidades geográficas.

Espacio de trabajo informativo ("Informative workspace")

Esta práctica nos dice que no hay nada más informativo que un tablón de corcho con notas de papel. El uso de sofisticadas herramientas para llevar el control de qué tareas están completas y qué tareas están pendientes.

La razón para esto es que, a pesar de que las herramientas informatizadas permiten tener teóricamente un mejor control de las tareas, la realidad es que estas tareas se hacen invisibles. Al tener todas las tareas pendientes y completas en un lugar visible del espacio de trabajo, se tiene una idea clara, a simple vista, sin necesidad de teclear nada, de qué se puede hacer ahora. El tener algo físico y tangible para llevar un control de progreso ayuda a que el objeto del trabajo aparene ser menos intangible.

¿Cómo se puede aplicar este principio a la cartografía?

La respuesta debería estar bastante clara: mapas de situación. Si se decide trabajar por cuadrícula, entonces es conveniente imprimir esa cuadrícula a gran tamaño, colgarla en la pared, y escribir sobre ella según se vayan completando cuadrantes o se vayan detectando zonas que necesitan corrección. O bien plastificarlo y utilizar rotuladores de pizarra blanca, o notas adhesivas. El objetivo es poder ver de un vistazo qué necesita atención, pero sin necesidad de lanzar un SIG, una aplicación o ver un documento en un ordenador.

Programación por pares ("Pair programming")

La programación por pares consiste en que haya un sólo teclado y monitor por cada pareja de trabajadores. Esto, que a priori parece contraproducente, tiene múltiples implicaciones positivas.

La primera es la transferencia de conocimiento. Al trabajar por parejas, ambas personas estarán constantemente hablando entre sí, intercambiando pequeñas técnicas, resolviendo dudas, o sugiriendo maneras liberamente distintas de trabajar. Como una parte de la programación por parejas es rotar la composición de las parejas cada poco tiempo (un par de horas), se consigue que exista una transferencia de conocimiento entre todo el equipo al completo.

La segunda ventaja de la programación por parejas es que el trabajo de alguien *siempre* está verificado por otra persona. En el ámbito del software, esto hace que la tasa de defectos sea menor, al estar constantemente verificando el trabajo de los demás y detectar muy rápidamente pequeños errores que pudieran pasar inadvertidos. El detectar los errores rápidamente es una parte importante de las ingenierías "intangibles", puesto que si pasan inadvertidos a otra fase, suponen un coste (tanto económico como de horas de trabajo) mucho más alto que el eliminar estos errores nada más se producen.

¿Cómo se puede aplicar este principio a la cartografía?

Este principio ha sido puesto en marcha, quizás inadvertidamente o por necesidad, por las grandes empresas de cartografía digital de redes de transporte. Lo habitual en estas compañías es realizar el trabajo de campo por parejas, en un coche: una persona conduce mientras otra digitaliza.

Esto hace que, al encontrar alguna entidad potencialmente conflictiva, se tengan de manera inmediata dos puntos de vista: se comprueba la entidad por una segunda persona sin necesidad de esperar a una fase de verificación y sin necesidad de enviar a esta segunda persona d nuevo a campo. Y, obviamente, si se rotan las parejas de trabajo de campo, al cabo de un tiempo todas habrán aprendido de la experiencia de los demás.

Integración continua ("Continous integration")

La integración contínua consiste en que desde que se realiza un cambio en el código hasta que ese cambio tiene repercusión en el producto final deben pasar como mucho unas horas, y este proceso debe ser totalmente automatizado y controlado.

En muchos sistemas informáticos, existen procesos que deben ser lanzados manualmente de manera sistemática. En bastantes ocasiones, el resolver un proceso manual toma tan sólo un poco menos de esfuerzo de lo que sería necesario para lograr automatizarlo completamente.

El objetivo de la integración contínua es establecer los mecanismos necesarios para que, cuando se añada o modifique código, el software se compile y despliegue automáticamente.

¿Cómo se puede aplicar este principio a la cartografía?

Cada vez que se añade más detalle, o se corrige un conjunto de datos, todo lo que dependa de ese conjunto de datos debe ser actualizado y desplegado en consonancia, de manera totalmente automática y contínua.

El mejor ejemplo de de integración contínua son todos los derivados de OpenStreetMap que se despliegan automáticamente con los datos máas actualizados. Actualmente, desde que un usuario modifica los datos hasta que se despliegan en la web pasan entre unos minutos y unas pocas horas. El proceso es relativamente complejo (exportar deltas de la BDD, comprimir, descargar, descomprimir, aplicar deltas a una BDD de sólo lectura, limpiar caché de teselas sucias a re-renderizar, renderizar y servir en tiempo real según demanda de la web), pero absolutamente nadie se tiene que preocupar de ninguno de los pasos. Una vez que se sabe cómo se tiene que trabajar con los datos, el coste extra de automatizarlo por completo es relativamente pequeño y hace que la carga de trabajo baje enormemente.

Programación con los tests por delante ("Test-first programming")

En la programación con los tests por delante, primero se realiza la batería de pruebas, y sólo entonces, cuando la batería de pruebas es funcional, se empieza la implementación.

Esto permite asegurarse de que el código funciona exactamente igual que como está previsto que funcione, o bien comprueba que su calidad es correcta. Por otro lado, permite ver si cualquier cambio realizado rompe la funcionalidad ya existente.

Lo más chocante de esta práctica es que es normal que las pruebas fallen al principio (puesto que se han realizado antes de aquello que intentan comprobar). Esto, que puede parecer contraproducente, realmente ayuda a saber cuándo el software puede considerarse como que está completo en algún aspecto.

¿Cómo se puede aplicar este principio a la cartografía?

En proyectos de SIG, se pueden realizar primero las herramientas que midan la calidad de los datos, y después (y/o mientras), crear los datos.

Por ejemplo: se pueden crear herramientas para medir cuán completo es el conjunto de datos, o automatizar las herramientas de comprobación topológica.

Por un parte, tener estas pruebas antes que los datos ayuda a tener una métrica de cuán completos o correctos son los datos. Por otra, el tener pruebas automatizadas permite detectar cuándo un cambio rompe otros datos que a priori no deberían estar afectados por los cambios.

En OpenStreetMap podemos encontrar al menos dos buenos ejemplos de esta práctica. Uno es el *coastline checker*: una vez cada pocos días comprueba de manera automática que la línea de costa de todo el mundo consiste en un polígono cerrado convenienemene, y avisa de los errores topológicos que pudiera haber. Así, si alguien rompe la línea de costa inintencionalmente, puede saberse cuándo y donde está el error. La clave está en que no se necesita una línea de costa totalmente completa antes de implementar esta herramienta.

Otro ejemplo es el *50 cities*, que sirve para comprobar la calidad de la topología de la red de carreteras. Consiste en, dadas 50 grandes ciudades norteamericanas, calcular la ruta mínima de la una a la otra y viceversa, y comprobar la longitud de esas rutas con otra longitud calculada a priori. La primera vez que se puso en marcha esta comprobación, muchas rutas entre muchas ciudades mostraban valores incorrectos (el objetivo de la programación con los tests por delante). A partir de entonces, el trabajo de comprobación topológica de la red se centró en aquellos tramos que hacían fallar las pruebas.

El tener pruebas antes que datos también es reconfortante para las personas trabajando en el proyecto. Dan una medida tangible de progreso, e incluso ayudan a reducir la ansiedad, porque permiten ver si cambios de última hora, potencialmente peligrosos, son demasiado disruptivos o no.

Diseño incremental ("Incremental design")

El principio de diseño incremental nos dice que no es necesario diseñar el sistema completamente antes de ponerse a construirlo. Esto es posible hacerlo con el software dado que es intangible, y corregir un fallo de diseño no implica volver a rehacer completamente lo que se ha hecho después.

Particularmente en el software, es bastante común el encontrarse con nuevos requisitos del cliente a mitad de un proyecto. Si la metodología dice que hay que hacer primero el diseño y después la implementación y despliegue, el equipo tendrá que rehacer el diseño, rehacer la implementación, y rehacer el despliegue. Por el contrario, si se da

por hecho que el diseño puede cambiar constantemente, todo estará preparado para acoger cambios de diseño sin necesidad de rehacer todo el sistema.

¿Cómo se puede aplicar este principio a la cartografía?

Fácil: no suponiendo que la estructura de la información geográfica es fija e inmutable. Es posible que, a mitad de un proyecto, se vea que las entidades necesitan una estructura distinta. En vez de rehacer el diseño de la base de datos geográfica (u otras estructuras de datos), y después volver a digitalizar toda la información, los sistemas deben estar preparados para poder modificar la estructura de la información geográfica. Asimismo, el equipo de trabajo debe saber cómo transformar la estructura sin destruir la información.

Un ejemplo de un sistema que admite diseño incremental es, de nuevo, OpenStreetMap. En OSM las entidades geográficas no tienen un número fijo y estático de campos para sus atributos, sino que tienen un número variable de etiquetas, y la definición de estas mismas etiquetas es también variable, incluso volátil en algunos casos.

Esto, que puede verse como una falta de planificación, permite que el diseño correcto se alcance únicamente con el paso del tiempo. La mayor ventaja es que permite una adaptación increíblemente rápida a nuevos tipos de entidades no planificadas a priori, como se demostró con el modelo de datos para el terremoto de Haití: añadir las características de edificio derruído y corte de carretera por escombros apenas llevó un par de horas, y no interfirió en absoluto con el resto de entidades.

Código compartido ("Shared code")

El principio de código compartido dice que no hay personas responsables de partes del código: todos son responsables de todas las partes.

Más concretamente, significa que si es necesario cambiar algo para poder corregir otra cosa distinta, no hay que pedir permiso a nadie. Sencillamente se cambia.

Este principio puede parecer contraproducente, pues parece que deja de haber personas responsables, o que habrá conflictos de responsabilidad sin que haya herramientas para evitarlos. En realidad, el resto de las prácticas (integración contínua, tests) hace que la responsabilidad compartida deje de ser un problema.

¿Cómo se puede aplicar este principio a la cartografía?

Actualmente, es común encontrar empresas o administraciones públicas en las que varios departamentos o secciones utilizan SIGs, y cada uno de estos departamentos es el responsable único de sus datos. Esto, que parece tener sentido a priori, es perjudicial a largo plazo. Muchos profesionales del SIG saben que en grandes empresas y administraciones, muchos retrasos y errores son debidos a que un departamento no tiene ningún acceso a los datos de otro departamento, y mucho menos puede corregirlos.

La solución es que cualquier persona pueda modificar los datos que haya realizado cualquier otra persona, aunque a priori no sean de su competencia ni tenga gran experiencia modificándolos. En el mejor de los casos, se ahorra trabajo a las dos partes y se acelera la transferencia de información correcta. En el peor de los casos, el resto de prácticas (como los tests por delante y la integración contínua) permitirán notificar rápidamente de cualquier error a las partes interesadas.

Dicho de otra manera, si alguien toca donde no tiene que tocar, es que tiene una razón para hacerlo, probablemente porque el territorio haya sufrido algún cambio. Al cambiar la información y provocar errores en los sistemas automáticos de validación, no se está destruyendo la integridad, sino haciendo notar que el territorio ha cambiado y el resto de las partes ha de actuar en consecuencia.

En definitiva, si lo que se busca es una mayor corrección de la información, el prohibir a otras personas dentro de la organización el modificarla es contraproducente.

Base de código única ("Single code base")

La práctica de la base de código única dice que el código del software debe estar centralizado en un sólo punto. Las personas y organizaciones sin experiencia en la gestión del desarrollo de software comúnmente caen en el error de tener varias copias ligeramente distintas del código en distintos ordenadores, sin tener idea de cuál de ellas es la correcta.

Para la gestión del software es frecuente utilizar sistemas de versiones como CVS (Concurrent Versioning System), SVN (SubVersion), GIT, Mercurial u otros. Estos sistemas llevan cuenta de quién es el autor de cada línea de código, de cuándo y porqué se han hecho todas y cada una de las modificaciones, y en qué consiste exactamente una modificación.

¿Cómo se puede aplicar este principio a la cartografía?

Si se puede detectar a un desarrollador novato por copias ligeramente distintas del código sin versionar, a alguien novato en SIG se le detecta por el rastro de *shapefiles*.

La práctica de la base de código única se traduce en tener una base de datos geográficos única. No debe haber una copia en cada ordenador.

Sin embargo, si bien existen paquetes SIG que permiten la gestión centralizada de bases de datos, y los controles de cambios, estas funcionalidades no se acercan a las que tienen los sistemas de control de versiones de código. Los sistemas de versionado existentes en la industria están orientados a poner tiritas a los flujos de trabajo existentes, pero no a dar pie a un flujo de trabajo con las ventajas de un versionado real. Es difícil saber, por ejemplo, la lista de personas que han editado una determinada entidad (mientras que es trivial saber quién ha tocado una cierta línea de código en un sistema de versionado de código).

Una base de datos geográficos única, con un versionado en condiciones, tiene multitud de ventajas. Sin embargo, el estado del arte de estas bases de datos deja todavía bastante que desear. WFS-V puede ser un paso en la dirección correcta, pero todavía necesita bastante esfuerzo para ser completamente funcional.

Un ejemplo de una base de datos geográficos que es única y funciona debidamente es OpenStreetMap. Existe un único servidor de datos, que controla versiones y usuarios sin ningún problema. Un interesante campo de estudio sería la utilización de esta tecnología para gestionar información gubernamental de manera conjunta entre varios organismos (implementando así los principios tanto de base única como de responsabilidad compartida).

CONCLUSIÓN

Varias de las técnicas de desarrollo ágil y programación extrema son aplicables al campo de la cartografía. Aunque estas técnicas no sean todavía muy utilizadas en el campo de los SIGs, merecen ser estudiadas para una posible implantación. Un caso de éxito de la implantación (inintencionada) de estas técnicas es OpenStreetMap, de donde se puede ver que estas prácticas y técnicas son válidas, si bien no existen datos que garanticen que su implantación es válida en cualquier entorno.

Sea como fuere, la recomendación del autor es que estas técnicas sean consideradas y evaluadas a la hora de realizar proyectos de SIG.

AGRADECIMIENTOS

A Javier Sánchez y a Micho, por su conocimiento de técnicas ágiles.

REFERENCIAS BIBLIOGRÁFICAS

Kent Beck, Cynthia Andres (2004): *Extreme Programming Explained: Embrace Change, 2nd* edition. Addison Wesley Professional. ISBN 0-321-27865-8.