Un nuovo metodo di Sviluppo Software: eXtreme Programming (XP)

César F. Acebal, Juan M. Cueva Lovelle
Edizione italiana a cura di ALSI e Tecnoteca
http://upgrade.tecnoteca.it
Traduzione di Cristina Tomacelli

Sintesi

Che cos'è l'eXtreme Programming, anche nota come XP ? Lo scopo di questo articolo è di rispondere a questa domanda, e di rivelare al lettore non esperto dell'argomento la natura di questo nuovo metodo di sviluppo del software. Naturalmente la lunghezza di un qualunque articolo tecnico non permette più di una breve introduzione a un qualunque nuovo metodo o tecnica, ma cercheremo di essere sufficientemente informativi, in modo che tutti voi andrete via con un'idea dei principi di base e per coloro che vorranno approfondire l'argomento, forniremo i riferimenti adatti.

Keywords: eXtreme Programming, XP, Software Development.

Introduzione

Come abbiamo detto nella parte iniziale e come indica anche il sottotitolo di questo articolo, XP è una nuova disciplina di sviluppo del software, che in mezzo al clamore delle fanfare, si è unita ai numerosi metodi e tecniche già esistenti. Per essere più precisi, questo è un metodo "leggero", in contrapposizione a un metodo "pesante" come Métrica. Prima di proseguire, vorremmo anche chiarire che in questo articolo faremo riferimento a XP come a un "metodo", a differenza della tendenza

ufficiale nell'IT di applicare il termine "metodologia" (scienza dei metodi) a ciò che non è nulla più che un insieme di metodi[1] o addirittura mere notazioni grafiche.

Si potrebbe dire che XP sia nato ufficialmente cinque anni fa in un progetto sviluppato da **Kent Beck** in *Daimler Chrysle*, dopo aver lavorato per parecchi anni con **Ward Cunningham** alla ricerca di un nuovo approccio al problema dello sviluppo del software che renda le cose più semplici degli attuali metodi a cui siamo abituati. Per molte persone, XP non è nulla di più del "buon senso comune". Perché dunque esso fa nascere così tante controversie e perché alcune persone lo adorano mentre molte altre lo disprezzano ? Come Kent Beck suggerisce nel suo libro [1], è possibile che sia perché XP porta avanti in modo estremo tecniche e principi del buon senso comune. Le tecniche più importanti sono:

- Il codice è costantemente rivisto, tramite il **pair programming** (due persone per macchina)
- I test sono fatti in ogni momento, non solo al termine di ogni classe (**test di unità**), ma anche i clienti che hanno bisogno di controllare che il progetto stia soddisfacendo i loro requisiti (**test funzionali**)
- I test di integrazione sono eseguiti sempre prima di aggiungere una qualsiasi nuova classe al progetto, o dopo averne modificata una già esistente (**integrazione continua**), utilizzando i *testing frameworks,* come ad es. *xUnit*
- (Ri)progettiamo ogni volta (**refactoring)**, cercando ogni volta di lasciare il codice nel più semplice stato possibile.
- Le iterazioni sono radicalmente più corte che negli altri metodi, in questo modo possiamo beneficiare dei commenti di ritorno il più spesso possibile.

Per riassumere, concludere questa sezione e proseguire andando nel dettaglio, vi lascerò con questa citazione dal già menzionato libro di Beck:

> *"Tutto cambia nel software. I requisiti cambiano. La progettazione cambia. Gli aspetti commerciali cambiano. La tecnologia cambia. I componenti del team cambiano. Il problema non è il cambiamento, di per sé, perché i cambiamenti avverranno; il problema, piuttosto, è l'inabilità di far fronte ai cambiamenti quando essi avvengono.*

---

[1] Ricardo Devis Botella. C++. STL, Plantillas, Excepciones, Roles y Objetos (Template, Eccezioni, Ruoli ed Oggetti). Paraninfo, 1997. ISBN 84- 283-2362-3

Le quattro variabili

XP precisa quattro variabili per un qualunque progetto: *costo, tempo, qualità* e *portata*.

Inoltre specifica che di queste quattro variabili, solo tre possono essere stabilite da elementi al di fuori del progetto (clienti e project manager), mentre il valore della variabile libera verrà stabilito dal gruppo di sviluppo in accordo con gli altri tre valori. Cosa c'è di nuovo in questo ? C'è che normalmente clienti e project manager considerano come loro lavoro quello di prestabilire il valore di *tutte* le variabili: "*Voglio che questi requisiti siano soddisfatti per i primi del mese prossimo, e devi fare in modo che la tua squadra lavori con questo obiettivo. (Oh, e tu sai che la qualità è la priorità numero uno !)*"

Naturalmente quando questo succede – e sfortunatamente succede piuttosto spesso – la qualità è la prima cosa che viene buttata dalla finestra. E questo accade per una semplice ragione che viene frequentemente ignorata: nessuno è in grado di lavorare bene quando è messo sotto una forte pressione.

XP rende visibili a tutti – programmatori, clienti e project manager - le quattro variabili; in questo modo i valori iniziali possono essere manipolati finché il quarto valore soddisfa tutti (naturalmente, con la possibilità di scegliere differenti variabili da controllare).

Inoltre le quattro variabili non sono in così stretta relazione come la gente normalmente si aspetta che siano. C'è un modo di dire molto conosciuto che recita: "nove donne non possono fare un bambino in un mese" che è applicabile in questo contesto. XP si focalizza su gruppi di sviluppo piccoli (dieci o dodici persone al massimo) che naturalmente possono essere ingranditi se necessario, ma non prima, altrimenti il risultato sarà generalmente l'opposto di ciò che si intendeva. Tuttavia, un certo numero di project manager sembra essere inconsapevole di ciò quando dichiarano, con orgoglio, che il *loro* progetto coinvolge 150 persone, come se ciò fosse un segno di prestigio, qualcosa da aggiungere al loro CV. E' buona norma, invece, aumentare il **costo** del progetto su aspetti come macchine più veloci, più specialisti in certe aree o uffici migliori per il gruppo di sviluppo.

Anche con la **qualità** avviene un fenomeno strano: spesso, aumentare la qualità significa completare il progetto in meno tempo. Il fatto è che appena il gruppo di progetto si abitua a fare test intensivi (e arriveremo presto a questo punto, in quanto è la pietra miliare di XP) e gli standard di codifica vengono seguiti, gradualmente il progetto inizierà a progredire più velocemente di quanto facesse prima. La qualità del progetto rimarrà assicurata al 100% - grazie ai test – e questo a sua volta instillerà maggiore confidenza nel codice e, quindi, maggiore facilità nel far fronte al cambiamento, senza stress, e questo permetterà alle persone di programmare più velocemente … e così via.

L'altra faccia della medaglia è la tentazione di sacrificare la qualità interna del progetto – quella percepita dai programmatori – per ridurre il tempo di rilascio del progetto, ritenendo che la qualità esterna, quella percepita dai clienti – non sarà eccessivamente deteriorata. Tuttavia, questa è una scommessa a brevissimo termine, che tende ad essere un invito al disastro, perché ignora il fatto basilare *che ognuno lavora meglio se gli è permesso di fare un lavoro di qualità*. Ignorare questo renderà il gruppo demoralizzato e, nel lungo termine, il progetto rallenterà, e verrà perso molto più tempo di quello che si sperava di guadagnare tagliando sulla qualità.

Per quanto riguarda la portata del progetto, è una buona idea lasciarla come variabile libera, in modo che, una volta fissate le altre tre variabili, il gruppo di sviluppo potrà decidere la portata in termini di:

- Stima delle attività da svolgere per soddisfare i requisiti del cliente.
- Realizzazione anticipata dei requisiti più importanti, in modo che in ogni momento il progetto offra la maggior funzionalità possibile.

Il costo del cambiamento

Anche se non possiamo qui approfondire dettagliatamente questo argomento, crediamo che sia importante almeno citare una delle più importanti e innovative supposizioni che XP fa in contrasto con la maggior parte dei metodi conosciuti. Ci riferiamo al costo del cambiamento. E' sempre stato considerato come verità universale che il costo del cambiamento nello sviluppo di un progetto cresca esponenzialmente nel tempo, come mostrato in figura 1:

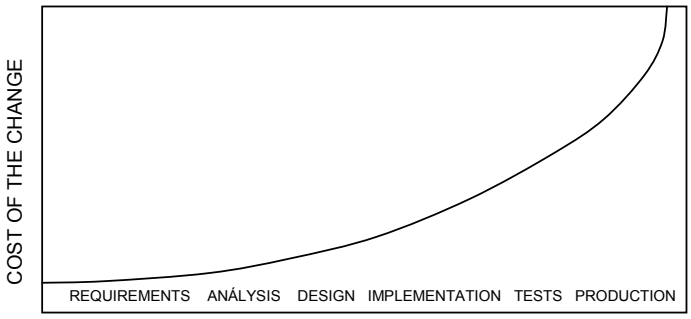

Figura 1. Costo del cambiamento nell'ingegneria del software "tradizionale"

XP dichiara che questa curva non è più valida, e che con una combinazione di buone pratiche di programmazione e di tecnologia sia possibile invertire la curva, come mostrato in figura 2:

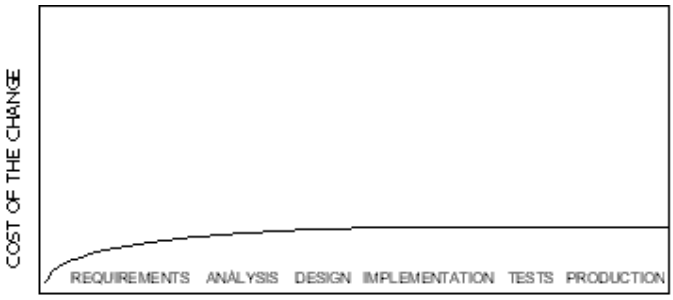

Figura 2. Costo del cambiamento in XP

Naturalmente, non tutti sono d'accordo con questa supposizione (e nel sito di Ron Jeffries [4] si possono leggere numerose opinioni di questo tipo). Ma in ogni caso è chiaro che se si decide di usare XP come processo di sviluppo software occorre accettare questa curva come valida.

L'idea di base è che invece di cambiare a causa dei cambiamenti, progettiamo nel modo più semplice possibile, facendo solo ciò che assolutamente necessario in un certo momento. Grazie all'estrema semplicità del codice, insieme alla nostra conoscenza del refactoring [5] e, soprattutto, il test e l'integrazione continua, i cambiamenti possono essere portati avanti ogni volta che è necessario.

Pratiche

Ma andiamo più nel dettaglio. Che cosa richiede effettivamente XP ? Cosa sono esattamente queste pratiche a cui abbiamo fatto riferimento, che sarebbero capaci di portare questo cambiamento di mentalità nello sviluppo del software ? Confidando nel fatto che tu, lettore, ci scuserai per la forzata brevità della nostra spiegazione, daremo ora una breve descrizione di queste pratiche.

Pianificazione

XP considera la pianificazione come un dialogo permanente tra la parte commerciale e tecnica coinvolte nel progetto, in cui la prima decide la portata – cosa è realmente essenziale per il progetto -, la priorità – cosa deve essere fatto prima -, la composizione dei rilasci – cosa dovrebbe essere incluso in ognuno – e le scadenze per ciascun rilascio.

I tecnici, per la loro parte, sono responsabili della stima del tempo necessario per realizzare le funzionalità che il cliente richiede, della messa in evidenza delle conseguenze delle decisioni prese, dell'organizzazione del lavoro e, infine, della pianificazione di dettaglio di ciascun rilascio.

Rilasci piccoli

Il sistema va in produzione inizialmente al più pochi mesi prima che sia completamente finito. I successivi rilasci saranno più frequenti – a intervalli di frequenza tra un giorno e un mese – il cliente e il gruppo di sviluppo beneficeranno dei ritorni prodotti da un sistema in produzione e questo si rifletterà sui successivi rilasci.

Progettazione semplice

Invece di impazzire nel produrre una progettazione che richieda il dono della chiaroveggenza per lo sviluppo, XP indica, in ogni momento, di progettare per le necessità del presente.

Testing

Qualunque caratteristica di un programma per la quale non c'è un test automatico semplicemente non esiste. Questa è indubbiamente la pietra miliare su cui è costruito XP. Altri principi possono essere adattati alle caratteristiche del progetto, all'organizzazione, al gruppo di sviluppo … Ma su questo punto non ci sono questioni: se non stiamo facendo test, non stiamo utilizzando XP. Dovremmo utilizzare qualche framework per testing automatico per fare ciò, come ad es. JUnit [9] o una delle sue versioni per differenti linguaggi.

Non solo, ma scriveremo i test prima ancora della classe che deve essere verificata. Questo è un aiuto per seguire il principio di *programmare per intenzione*, cioè, scrivere il codice come se i metodi più costosi fossero già stati scritti, cosicché dovremmo mandare solo i corrispondenti messaggi, in modo che il codice sia veramente il riflesso della sua "intenzione" e si autodocumenterà. Nel sito di Junit, citato nel precedente paragrafo, potete trovare degli interessanti articoli che spiegano come scrivere questi test.

Refactoring

Questa pratica risponde al principio di semplicità e di fatto consiste nel lasciare il codice esistente nel più semplice stato possibile, in modo che non sia persa nessuna funzionalità – o guadagnata – e tutti i test continuino ad essere effettuati correttamente. Questo ci renderà più confidenti nel codice già scritto e quindi meno riluttanti nel modificarlo quando alcune caratteristiche dovranno essere aggiunte o modificate. Nel caso di sistemi legacy, o progetti presi in carico dopo essere già partiti, potrebbe capitare la necessità di dedicare alcune settimane esclusivamente al refactoring del codice – cosa che tende a essere una fonte di tensione con i project manager coinvolti, quando viene detto loro che il progetto deve essere tenuto fermo per parecchi giorni "solo" per modificare codice esistente, che funziona, senza aggiungere nuove funzionalità ad esso.

Programmazione in coppia (Pair programming)

Tutto il codice sarà sviluppato in coppia – due persone che condividono un solo monitor e una sola tastiera -. La persona che scrive il codice penserà al miglior modo per realizzare un particolare metodo, mentre il suo collega farà lo stesso, ma da un punto di vista più strategico:

- Stiamo facendo nel modo giusto ?
- Cosa potrebbe andare storto qui ? Cosa dovremmo controllare nei test ?
- C'è un modo per semplificare il sistema ?

Naturalmente i ruoli sono interscambiabili, in modo che in ogni momento la persona che osserva possa prendere possesso della tastiera per dimostrare un'idea o semplicemente per far riposare il suo collega. Allo stesso modo la composizione delle coppie potrebbe cambiare ogni volta che uno dei due fosse richiesto da un altro membro del gruppo per dargli una mano con il suo codice.

Proprietà collettiva del codice

Chiunque può modificare qualunque parte del codice, in un momento qualsiasi. In realtà, chiunque veda un'opportunità per semplificare, facendo refactoring, una qualunque classe o metodo, indipendentemente dal fatto che l'abbia creata oppure no, non dovrebbe esitare a farlo. Questo non è un problema in XP, grazie all'uso di standard di codifica e all'assicurazione che il test ci dà in merito al fatto che tutto continuerà a funzionare dopo ogni modifica.

Integrazione continua

Ogni poche ore – o verso la fine di ogni giorno di programmazione – il sistema completo è integrato. Per questo scopo c'è ciò che è noto come macchina di integrazione, alla quale andrà ogni coppia di programmatori quando essi avranno una classe che ha superato il test di unità. Se dopo aver aggiunto la nuova classe insieme ai suoi test di unità il sistema completo continuerà a funzionare – ovvero, supererà tutti i test -, i programmatori considereranno la loro attività completata. Altrimenti essi saranno responsabili per riportare indietro il sistema in uno stato in cui tutto funzionava al 100%. Se dopo un certo periodo di tempo non saranno in grado di scoprire cosa non va, dovranno gettare via il loro codice e ricominciare da capo.

40 ore alla settimana

Se veramente vogliamo offrire qualità, e non semplicemente un sistema che funziona – che come tutti sappiamo, in IT, è un problema triviale[2] - vorremo che ogni membro del nostro gruppo si alzi al mattino riposato e vada a casa alla sera alle 6, stanco ma con la soddisfazione di un lavoro ben fatto, e quando arriva il venerdì egli o essa possa avere davanti due giorni di riposo da dedicare a cose che non hanno nulla a che fare con il lavoro. Naturalmente non devono essere per forza 40 ore – potrebbero essere tra le 35 e le 45 - , ma una cosa è certa: nessuno è capace di produrre lavoro di qualità in 60 ore alla settimana.

Cliente sul posto

Un'altra regola controversa di XP: almeno un cliente reale dovrebbe essere permanentemente disponibile al gruppo di progetto per rispondere a qualunque domanda i programmatori possano avere per lui, per stabilire priorità … Se il cliente controbatte che il suo tempo è troppo prezioso, dovremmo aver presente che il progetto che ci è stato dato è così triviale che il cliente non lo considera degno della sua attenzione, e che quindi non gli importa se esso è basato su supposizioni fatte dai programmatori che sanno poco o nulla sul reale funzionamento del business del cliente.

Standard di codifica

Essi sono essenziali per il successo delle proprietà collettiva del codice. Questo sarebbe impensabile senza una codifica basata su standard che permettano a chiunque di sentirsi a proprio agio con il codice scritto da altri membri del gruppo.

---

[2] Ricardo Devis Botella. *Curso de Experto Universitario en Integración de Aplicaciones Internet mediante Java y XML*. (Corso di Specialista Universitario nell'Integrazione di Applicazioni Internet mediante Java e XML) Università di Oviedo, 2000.

Pianificazione

Anche se XP è incentrato sul codice non è solo questo. E' anche e soprattutto un metodo di project management, a dispetto delle critiche sollevate da molte persone, forse dopo una troppo affrettata lettura di un articolo come questo. Ma per chiunque abbia avuto la buona volontà di leggere uno dei libri che spiegano il processo, risulta chiaro che la pianificazione è una parte fondamentale di XP. Il punto è che, assunto che lo sviluppo del software, come la maggior parte delle cose in questa vita, è un processo caotico, XP non tenta di trovare un determinismo inesistente ma piuttosto fornisce i mezzi necessari per trattare questa complessità, e la accetta, senza cercare di forzarla in vincoli di pesanti e burocratici metodi. Raccomandiamo caldamente la lettura della gentile introduzione alla teoria del caos di Antonio Escohotado [11], che pensiamo abbia molto a che fare con l'idea sottostante XP. In breve, i metodi leggeri – e XP si annovera in essi – sono adattativi piuttosto che predittivi [8].

Il ciclo di vita

Se, come è stato dimostrato, i lunghi cicli di sviluppo dei metodi tradizionali non sono in grado di far fronte al cambiamento, forse ciò che dovremmo fare è rendere i cicli di sviluppo più corti. Questa è un'altra delle idee centrali di XP. Diamo un'occhiata allo schema che confronta il modello a cascata, il modello a spirale e XP:

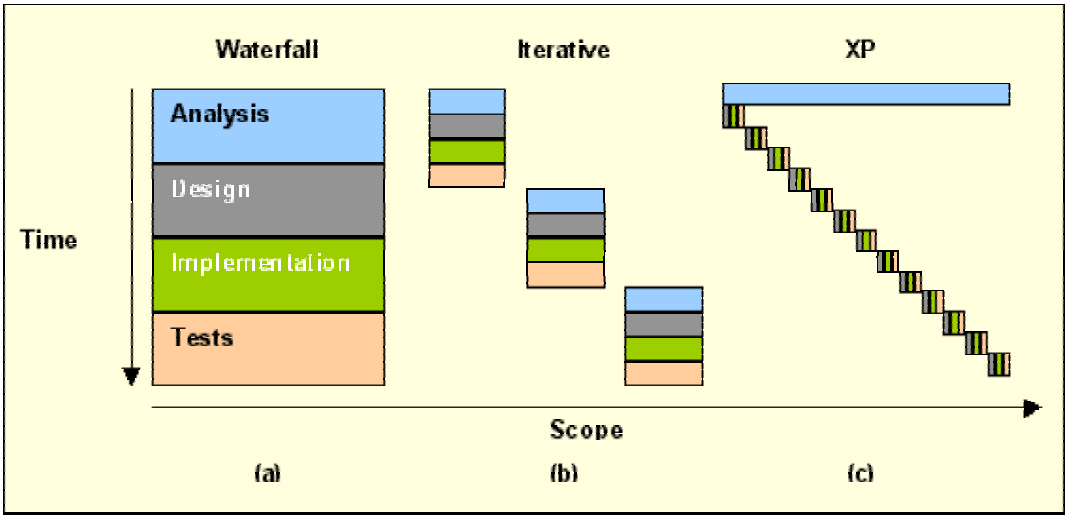

Figura 3 Confronto dei cicli di sviluppo lunghi: (a) il modello a cascata (b) i cicli iterativi più corti, per es., il modello a spirale e (c) l'insieme di tutte queste attività che XP impiega durante l'intero ciclo di sviluppo del software

Conclusioni

Come detto all'inizio dell'articolo, XP, dopo appena un anno dalla pubblicazione del primo libro sull'argomento, ha causato un gran furore all'interno della comunità di ingegneria del software. Il risultato dell'indagine riportata sotto, commissionata da IBM, evidenzia il fatto che le opinioni sull'argomento sono divise [10]:

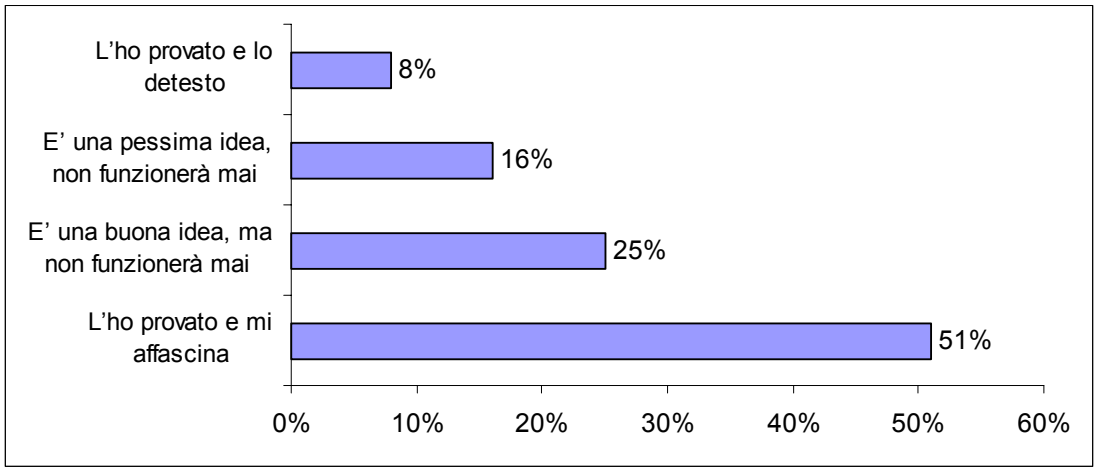

Figura 4. Indagine IBM (Ottobre 2000): Che cosa pensi dell'eXtreme Programming?

La programmazione in coppia è quella maggiormente colpita da forti critiche – soprattutto da parte dei project manager, anche se è un'opinione che è senza dubbio condivisa da molti programmatori che hanno un senso eccessivo di proprietà sul codice sviluppato ("L'ho fatto io, e in più sono così bravo a programmare e ho una così buona padronanza del linguaggio che solo io posso capirlo") -, ma molto viene detto anche riguardo al mito delle 40 ore alla settimana (?), ovvero "tutta questa faccenda riguardo ai test va benissimo se hai tempo da vendere, ma è un lusso che non ci si può permettere nelle attuali condizioni di mercato" … a molte altre critiche vigorose dello stesso tipo.

Ci sono anche persone che dicono, (e queste critiche sono forse più fondate delle precedenti) che XP funziona solo per buoni sviluppatori, ovvero, persone come Kent Beck, che sono in grado di progettare bene, in modo semplice, e allo stesso tempo – e probabilmente proprio per questo motivo – facilmente estensibile, fin dall'inizio.[3].

Una delle cose che stiamo cercando di dire è che XP non dovrebbe essere male interpretata a causa dell'inevitabile superficialità di articoli come quello che state leggendo. Alla fin fine l'ideatore di

---

[3] Raúl Izquierdo Castanedo. *Comunicación privada. (Comunicazione privata)*

questo metodo non è un novellino, ma uno dei pionieri nell'uso dei template software, ideatore dei file CRC, autore del framework per editor grafici HotDraw e del framewok per testing *xUnit.* Anche solo per questo, varrebbe la pena almeno di dare un'occhiata a questo nuovo ed eccitante metodo di sviluppo software.

Tuttavia, nessuna delle pratiche raccomandate da XP sono un'invenzione del metodo; esistevano tutte da prima, e ciò che ha fatto XP è stato di metterle insieme e di provare che funzionano.

In ogni caso, il primo libro di Beck è una boccata di aria fresca che dovrebbe essere obbligatorio leggere per qualunque ingegnere del software – o architetto del software, per usare il termine preferito dal nostro amico Ricardo Devis-, a qualunque conclusione voi arriviate riguardo a XP. Per lo meno, è molto divertente da leggere.

Riferimenti

[1] **Kent Beck.** *EXtreme Programming Explained: Embrace Change* Addison Wesley Longman, 2000. ISBN 201-61641-6

[2] **Ronald E. Jeffries et al.** *EXtreme Programming Installed* Addison-Wesley, 2000. ISBN 0201708426

[3] **Kent Beck, Martin Fowler.** *Planning EXtreme Programming* Addison-Wesley. ISBN 0201710919

[4] **www.xprogramming.com.** Uno dei portali XP più completi, di Ron Jeffries.

[5] **Martin Fowler.** *Refactoring : Improving the Design of Existing Code*. Addison-Wesley, 1999. ISBN 0201485672

[6] **Kent Beck.** *Embracing Change with EXtreme Programming*. Computer (rivista di *IEEE Computer Society*).Vol. 32, No. 10. October 1999, pp. 70-77

[7] **http://www.computer.org/seweb/Dynabook/Index.htm.** *eXtreme Programming. Pros and Cons. What questions remain?* Il primo libro "dinamico" che *IEEE Computer Society* ha dedicato a XP, con una serie di articoli correlati.

[8] **Martín Fowler.** *The New Methodology.* www.martinfowler.com/articles/newMethodology.html

[9] **www.junit.org.** JUnit, un framework per il test automatico in Java, adattato da un framework dello stesso tipo per Smalltalk, e disponibile in molti altri linguaggi. Queste altre versioni, sotto il nome generico di xUnit, sono disponibili sul sito web di Ron Jeffries [4], nella sezione software.

[10] **www-106.ibm.com/developerworks/java/library/java-pollresults/xp.html.** *Java poll results: What are your thoughts on EXtreme Programming?* Indagine IBM, Ottobre 2000

[11] **Antonio Escohotado.** *Caos y orden (Chaos and order).* Espasa Calpe, 1999. ISBN 84-239-9751-0. Una interessante introduzione alla teoria del caos, che secondo noi descrive l'attitudine di cui si ha bisogno per approcciare allo sviluppo software dal punto di vista di XP.

Autori

César Fernández Acebal ha conseguito la laurea in ingegneria informatica presso l'Università di Oviedo. Ha lavorato come insegnante di Java e di programmazione Web per gli studenti delle scuole superiori. Successivamente, ha lavorato come direttore tecnico in una società di sviluppo di siti web. Ha combinato queste posizioni con una continua attività di formazione relativa a Java, XML, sviluppo Web, ecc. E' attualmente un architetto IT della società di e-business B2B 2000. I suoi interessi di ricerca includono la programmazione object-oriented, l'ingegneria del software e i processi di "software agile". E' membro di ATI, IEEE, Computer Society, ACM, ecc. <acebal@ieee.org>

Juan Manuel Cueva Lovelle è un ingegnere minerario laureatosi nel 1983 nella scuola Tecnica di Ingegneri Minerari di Oviedo (Università di Oviedo). Ha ottenuto il dottorato di ricerca dall'Università Tecnica di Madrid nel 1990. Dal 1985 è professore di Linguaggi e Sistemi di Computer all'Università di Oviedo. E' membro votante di ACM e IEEE. I suoi interessi di ricerca includono la tecnologia Object-Oriented, Linguaggi dei Processori, Interfaccia Uomo-Macchina, Database Object-Oriented, Progettazione Web, Progettazione Object-Oriented, Metodologie Object-Oriented, XML, WAP, Modellazione Software con UML e Sistemi Informativi Geografici. <cueva@lsi.uniovi.es>