Extreme Programming e Metodologie Agili di Sviluppo Software:

concetti, prodotti e risorse

Pubblicato da Massimiliano Bigatti
max@bigatti.it
http://www.bigatti.it

Dicembre 2002

Contenuti

Abstract

Nel corso degli anni '90 è emerso un certo numero di metodologie di sviluppo software, cosiddette "agili". Un sinonimo per questa accezione della parola agile, potrebbe essere "leggero" (lightweight): queste metodologie si contrappongono infatti a quelle cosiddette pesanti (heavyweight), come la waterfall o quella iterativa, proponendo modalità di lavoro fortemente orientate ai risultati. La "leggerezza" di queste discipline è relativa all'impegno da profondere per il raggiungimento del risultato: invece che pianificare, progettare e sviluppare tutto il sistema software, che oltre alle funzionalità richieste cerchi anche di anticipare quelle future, le discipline agili si focalizzano invece sul raggiungimento di un risultato alla volta, piccolo e ben definito, costruendo con un processo iterativo il sistema completo.

In questo documento vengono descritti concetti, valori, principi, pratiche, strumenti e risorse delle discipline agili.

Metodologie pesanti

In una metodologia di sviluppo "pesante" come quella a cascata (waterfall - Figura 1), le fasi di pianificazione, progettazione, sviluppo e test sono eseguiti in forma sequenziale: per poter cominciare, ciascuna fase necessita la conclusione di quella precedente.

Figura 1

I problemi di questa metodologia sono evidenti: sebbene la semplice struttura del processo semplifichi il lavoro degli sviluppatori, spesso in una qualche fase ci si rende conto che sarebbe necessario tornare a quella precedente. Ad esempio, in fase di sviluppo, potrebbe risultare necessario tornare alla fase di analisi per chiarire meglio alcuni aspetti del sistema.
Inoltre questo modello non si sposa bene con la stragrande maggioranza dei progetti, dove i requisiti utente non sono scritti nella roccia, ma cambiano spesso. Questa metodologia

andava bene ad esempio per lo sviluppo di grossi software come i primi sistemi operativi, dove i requisiti potevano essere definiti a priori e l'esperienza di sviluppo del software non era ancora molta.
Per ovviare ai problemi del modello a cascata, è nata la metodologia iterativa (Figura 2), ancora oggi ampiamente usata.

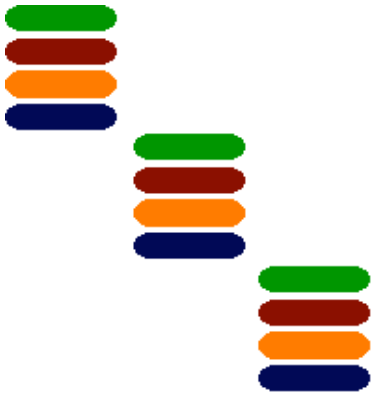

Figura 2

Nel modello iterativo sono presenti le stesse fasi del modello a cascata, ma i tempi sono più ristretti e dalla fase di testing si torna poi a quella di pianificazione per applicare eventuali correzioni al risultato dello sviluppo.

Metodologie agili

In contrapposizione alle metodologie cosiddette “pesanti”, dalla fine degli anni ’80 - inizio degli anni ’90, sono nate le prime discipline “agili”, focalizzate a risolvere ricorrenti problemi riscontrati nella conduzione di progetti software.
L’idea di base nelle metodologie agili è che queste non sono *predittive*, non cercano cioè di prevedere come evolverà il sistema software, ma sono *adattive*, propongono cioè valori e pratiche per meglio adattarsi all’evoluzione dei requisiti utente, prima che del sistema software [3]. Nei progetti software, le esigenze del cliente sono in costante mutamento: sebbene l’avere in anticipo tutti i requisiti utente sia un aspetto desiderabile, spesso non è ottenibile. In questi casi, non si può utilizzare una metodologia predittiva, perché il processo non è completamente prevedibile. Ma anche se fossero disponibili subito tutti i requisiti, ed anche se fossero tutti coerenti, anche la fase di progettazione potrebbe dare dei problemi: è molto difficile, ad esempio, catturare il sistema in un modello UML che sia completo e corretto e che possa essere utilizzato come base per lo sviluppo del codice. Spesso anche i progettisti si sorprendono di come un progetto viene poi tradotto in codice [3].
Visti questi problemi, le metodologie agili si concentrano dunque nel facilitare le modifiche al sistema, piuttosto che su

altri aspetti, quali produrre il sistema più efficiente possibile, o più riutilizzabile o più prestante.
In contrapposizione alle metodologie pesanti, quelle agili si basano su un processo iterativo composto da fasi di progettazione, sviluppo e test molto più brevi (Figura 3).

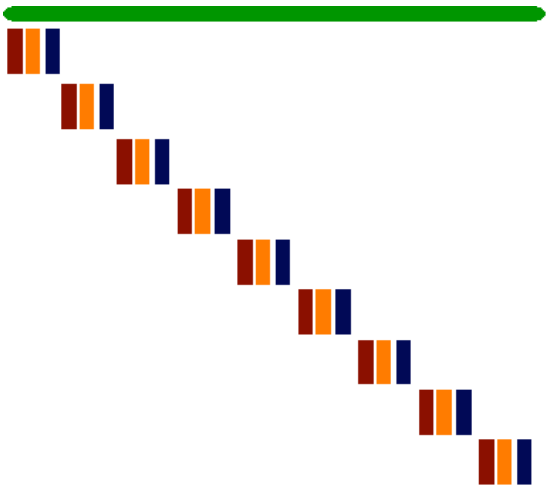

Figura 3

Tutte le metodologie agili condividono una serie di caratteristiche fondamentali:

adattabilità. Il progetto ed il codice sono impostati per privilegiare l'adattabilità alle modifiche, piuttosto che per soddisfare pienamente ed in modo pianificato tutte le specifiche. Sebbene progettare da subito tutto il sistema avendo da subito tutte le specifiche in modo completo, può

portare a progetti ben fatti, questa opzione si basa su un fondamento erroneo: non è possibile, o è estremamente difficile, ottenere a priori tutte le specifiche in modo completo;

iteratività ed incrementalità. Nelle metodologie di sviluppo agili, ed in particolare nell'Extreme Programming, le fasi di pianificazione, progettazione, sviluppo e test sono compresse in tempi molto più ridotti delle metodologie tradizionali, concentrandosi sulla soluzione di pochi problemi, piccoli e ben definiti;

rilasci frequenti. Grazie al procedimento composto da piccoli passi, il team di sviluppo è in grado di produrre versioni del software in tempi più ridotti. I rilasci risultano quindi più frequenti;

testing. Una delle pietre miliari delle discipline agili è il testing, la verifica automatica del corretto funzionamento del sistema. Questo si applica sia al codice, che ai dati, ed agli altri prodotti a cui le diverse discipline sono orientate (come i modelli o la documentazione). Dove non è possibile un test automatico, viene proposta una verifica manuale da parte di colleghi di pari esperienza;

orientamento alle persone. Nelle metodologie agili vengono privilegiate le persone invece dei processi: il concetto si basa sul fatto che se una persona è felice di lavorare, produrrà un sistema di qualità superiore ed in meno tempo, contrariamente, chi è obbligato a svolgere attività sgradite, avrà problemi di concetrazione, qualità e risultati;

derivano dal buon senso. Le pratiche ed i valori delle discipline agili non sono state inventate di sana pianta dai rispettivi autori, ma derivano dal buon senso comune, anche se vengono in certi casi portate all'estremo;

applicabili dove i requisiti cambiano velocemente. Le discipline agili non sono sempre applicabili a tutti i progetti: sono indicate in progetti dove i requisiti utente cambiano continuamente, ed i team di sviluppo ne deve seguire in tempo quasi reale l'evoluzione.

Una supposizione interessante delle metodologie agili, ed in particolare di XP, in contrasto con la maggior parte dei metodi conosciuti, è quella secondo cui, con il progredire del tempo, il costo del cambiamento non cresce, ma al contrario si stabilizza ad un valore modesto.

Variabili in gioco nei progetti software

Extreme Programming precisa quattro diverse variabili che entrano in gioco nei progetti software: *costo, tempo, qualità e portata*. L'elemento nuovo di XP è il fatto che solo tre di queste variabili sono definite a priori, mentre la quarta è il grado di libertà che sarà stabilita dal gruppo di lavoro in base al valore delle altre. Ad esempio, a parità di tempo e qualità, al crescere della portata, il gruppo di lavoro potrebbe determinare che il costo aumenta proporzionalmente; oppure a parità di costo e portata, se il management richiede un'alta qualità, il tempo potrebbe crescere di conseguenza.
Il management solitamente, tende invece a fissare a priori il valore di tutte e quattro le variabili, mettendo in difficoltà il gruppo di lavoro.
Sicuramente la variabile più problematica è la qualità: quella interna, percepita dagli sviluppatori, deve essere alta, in modo che, unitamente ai test, possa garantire una forte progressione nello sviluppo richiesta da XP. Il problema è, che quando non c'è tempo, la qualità è la prima cosa che viene accantonata, proprio perché nessuno è in grado di lavorare bene quando è messo sotto stress. Il fatto è che ognuno lavora meglio, se gli è permessso di fare un lavoro di qualità: in caso contrario il

morale potrebbe risentirne e di conseguenza la velocità di progressione del progetto.

Agile Alliance

Attorno ai valori ed ai concetti di base di tutte le discipline agili (vedi *Storia e panoramica delle discipline agili*), è nata un'alleanza di persone, la Agile Alliance [web4], che ha pubblicato i principi su cui si basa, in un "manifesto" [web5] disponibile su Internet.
Gli elementi chiave del manifesto sono:

le persone e le interazioni prima dei processi e strumenti. Conformemente a specifiche discipline, come l'Extreme Programming, il manifesto per lo sviluppo agile del software privilegia le persone e le interazioni tra di esse, cercando di ottenere il meglio da ciascuna persona, senza imbrigliarle in processi e con strumenti che ne possano limitare la creatività ed abbassare il morale;

software che funziona prima della documentazione comprensiva. Il movimento agile ritiene che per prima cosa sia necessario costruire un sistema software che funziona, e solo in un secondo tempo pensare alle problematiche di documentazione. La documentazione non deve quindi sopperire alle mancanza od ai malfunzionamenti del software;

collaborazione con i clienti al posto della negoziazione del contratto. Il movimento ritiene che sia di primaria importanza stabilire un rapporto di collaborazione con il cliente, in modo che sia anche lui una forza trainante nello sviluppo del progetto. Si ritiene che la collaborazione produca risultati migliori rispetto a pratiche che prevedano di imbrigliare il cliente all'interno di contratti limitanti e stringenti;

risposta ai cambiamenti piuttosto che aderenza alla pianificazione. Uno dei principi di base condivisi da tutte le discipline agili è l'adattabilità alle modifiche piuttosto che una predittività di difficile realizzazione. Tutte le pratiche proposte dalle discipline agili sono volte proprio a migliorare il modo con cui il gruppo di lavoro risponde ai cambiamenti dei requisiti e dell'ambiente circostante.

Storia e panoramica delle metodologie agili

Il movimento agile trova una forte promozione nel metodo XP, (Extreme Programming), nato probabilmente durante un progetto sviluppato da Kent Beck per Daimler Chrysle, anche coadiuvato dalla ricerca fatta negli anni passati congiuntamente a Ward Cunningham. Kent Beck è un esperto informatico: è stato uno dei pionieri, oltre che di XP, anche dell'uso dei template nel software, ha ideato i file CRC, il framework per editor grafici HotDraw e quello per il testing xUnit.

Nella Extreme Programming si inseriscono alcune pratiche, tra cui il refactoring del codice, formalizzato nel noto testo di Martin Fowler, "Refactoring: Improving the Design of Existing Code" [libro1]. Sulla base dei concetti di XP è nato SCRUM, un modo agile per gestire i progetti software, dove si ribadisce che il ciclo di sviluppo è composto da molte iterazioni di breve durata, in particolare con brevi meeting di dieci minuti ogni giorno e sprint produttivi di un mese. Tra XP e SCRUM si posizionano una serie di metodologie, come l'Agile Modeling, che si pone l'obiettivo di modellare e documentare i sistemi software puntando direttamente all'obbiettivo, oppure l'Agile Data, che porta i concetti agili nello sviluppo dei database,

anche proponendo un catalogo di refactoring per i dati.
Entrambe queste ultime discipline sono state sviluppate da Scott Ambler, già noto in passato per “The Object Primer” e “The Elements of Java Style”.
Degno di nota anche il Feature Driven Programming, una metodologia creata da Peter Coad e Jeff De Luca e pubblicata per la prima volta nel loro libro *Java Modeling in Color with UML* [libro4], che pone l’attenzione sulle funzionalità richieste dal cliente.
Altre medotologie meno note, ma che hanno contribuito significativamente al movimento delle discipline agili sono DSDM, ASD e Crystal.

Nelle prossime sezioni verranno affrontate più nel dettaglio alcune di queste metodologie.

Metodologie Agili

Extreme Programming

La disciplina che ha portato alla ribalta le metodologie agili è sicuramente la programmazione estrema (Extreme Programming [web1]). Sebbene questa disciplina sia molto nota per l'approccio alla progettazione e codifica, essa è primariamente un metodo di gestione del progetto, aspetto su cui si è concentrata l'evoluzione dell'ultimo periodo.
XP basa su valori di base e pratiche. I valori fondamentali sono:

comunicazione. In XP c'è una forte comunicazione tra tutti gli attori coinvolti: gli sviluppatori, il management ed il cliente;

semplicità. Il risultato deve essere semplice e risolvere il solo ambito del problema che è stato posto: non deve cercare di anticipare i problemi futuri ma concentrarsi sulla soluzione del problema contingente. Lo sviluppatore deve pensare in piccolo, concentrando lo sforzo in un ambito ristretto;

feedback. Il sistema è costantemente verificato, in modo da assicurarne la consistenza in ogni momento. Questo

avviene tramite la creazione di *test di unità* che possono essere eseguiti dalla macchina stessa.

coraggio. Se ci si rende conto sia necessario modificare quello che è stato fatto in precedenza, in prima persona o da un altro sviluppatore del team, è necessario procedere senza timore cambiando strada per rimanere vicini all'obiettivo.
Si noti che questi valori sono applicabili ai diversi aspetti dello sviluppo software: dalla gestione (management), alla progettazione, allo sviluppo, fino alla documentazione.
Oltre ai valori, XP si basa specifiche pratiche, tecniche che vogliono concretizzare i valori basilari della disciplina.
L'Extreme Programming prevede una trentina di pratiche, qui saranno descritte le più importanti:

pair programming. Extreme Programming prevede che per ogni postazione di lavoro siano presenti due persone. Lo scopo del secondo sviluppatore è quello di verificare che il primo sviluppatore scriva il codice in modo corretto ed eventualmente per sollevare potenziali problemi o proporre vie alternative di sviluppo;

testing. Prima di scrivere il codice dell'applicazione, viene scritto quello di test. Vengono identificati pochi ma sostanziali test che mettano in luce eventuali

malfunzionamenti del codice e se ne codifica l'implementazione. Una volta scritto il test, si passa allo sviluppo del codice vero e proprio. Un aspetto interessante è quello che quando si presenta un bug, viene creato subito un test per evidenziarlo. Si noti che il testing è un elemento basilare delle metodologie agili: in mancanza di questo non ci si trova in presenza di Extreme Programming;

refactoring. Il sistema è soggetto a continua riscrittura volta a lasciare il codice nel più semplice stato possibile che consenta l'implementazione delle funzionalità richieste. Nessuna funzionalità viene aggiunta prima che sia effettivamente necessaria;

standard e proprietà collettiva. La codifica avviene secondo standard definiti, per promuovere la proprietà collettiva del codice. Tutti gli sviluppatori sono responsabili dell'intero sistema e possono in qualsiasi momento intervenire per eseguire gli eventualmente necessari refactoring. Gli sviluppatori dovrebbero inoltre cambiare spesso collega e parte del progetto su cui lavorano, per essere aggiornati sul funzionamento delle altre parti del sistema ed eventualmente per acquisire skill differenti. Lo sviluppo avviene da parte della collettività, senza specializzazioni verticali;

rilasci piccoli e frequenti. I rilasci del software sono il risultato di iterazioni molto brevi, dove poche e piccole funzionalità vengono implementate nel sistema e subito rilasciate al cliente per il test di accettazione;

integrazione frequente e seriale. L'integrazione delle modifiche nel sistema viene fatta frequentemente, in modo da limitare i conflitti che il nuovo codice (ed il codice di test) dovesse dare. Inoltre, questa procedura avviene in modo strettamente sequenziale: solo uno sviluppatore alla volta integra le sue modifiche con il sistema, testa e rilascia il codice. Questo, assieme alla programmazione collettiva, consente di risolvere in modo semplice le tipiche problematiche di integrazione su una stessa base di codice del lavoro di diverse persone.

SCRUM

SCRUM è una disciplina agile di gestione del processo che funge da contenitore di pratiche di ingengeria del software già esistenti (come ad esempio le pratiche *di codifica* di XP).
Basato su un approccio iterativo ed incrementale, è studiato per sistemi dove i requisiti cambiano velocemente, fornendo dei modi per migliorare la comunicatività e massimizzare la cooperazione.
Si basa su un elenco di caratteristiche funzionali richieste (in modo similare a FDD) che vengono assegnate al team di sviluppo che se ne occuperà durante una iterazione di sviluppo composta da massimo 30 giorni e chiamata *sprint*.
All'interno di questa iterazione, sono presenti sotto-iterazioni di 24 ore, scandite da un meeting di inizio giornata, per far emergere eventuali ostacoli da gestire e per sincronizzare il team su quanto fatto nel giorno precedente e su quanto si intende fare nel giorno successivo. Alla fine dello sprint, il team di sviluppo fornisce la nuova funzionalità dimostrabile al cliente.

DSDM

DSDM, è uno standard europeo che si sta diffondendo sempre di più nell'ambiente. Il consorzio DSDM, che ne gestisce l'evoluzione, definisce DSDM come un framework, piuttosto che un metodo. Basato anche questo su sviluppo iterativo ed incrementale, prevede però fasi di pre-project, studio di fattibilità e studio funzionale che vengono eseguiti in modo seriale. Questo aspetto rende DSDM un ibrido agile-monolitico, in quanto "l'agilità" è limitata alle fasi di analisi funzionale, tecnica e di implementazione. Un aspetto importante, presente anche in altre discipline come l'Agile Modeling, è che in fase di studio di fattibilità, il metodo prevede di valutare se per il progetto specifico, questa disciplina è l'approccio corretto. Alcuni principi di base di DSDM condivisi con le altre discipline sono la collaborazione con il cliente, la consegna frequente di versioni funzionanti, integrazione del testing. Alcuni elementi peculiari che le altre discipline non sottolineano come DSDM, sono il coinvolgimento dell'utente finale ed il fatto che qualsiasi modifica durante lo sviluppo sia reversibile.

Agile Modeling

L'Agile Modeling (AM) è la metodologia proposta da Scott Ambler per la creazione di modelli agili dell'applicazione e per la documentazione. Il concetto di base di AM è che un modello agile è un modello *buono quanto basta*. Questo prevede che il modello, sebbene il più semplice possibile, contenga del valore aggiunto che sia chiaro, dettagliato, accurato, consistente e comprensibile. Si pone come complementare ad altre discipline, come XP, DSDM o SCRUM. In effetti, la sua focalizzazione sul modello e la documentazione lo rende parallelo a queste discipline, anche perché è in effetti orientato ad aspetti definiti come non primari nel manifesto dello sviluppo agile del software (vedi Agile Alliance).
La disciplina si basa sui concetti comuni alle metodologie agili, come la proprietà collettiva e la costruzione tramite piccoli passi. Inoltre, promuove la creazione di più modelli in parallelo e la verifica con codice concreto.

Agile Data

Sempre Ambler propone di applicare le metodologie agili ai dati, in particolare a quelli memorizzati in database relazionali. Questo è un problema concreto, se il codice è sviluppato in modo incrementale tramite una disciplina agile, per applicazioni che fanno uso di database (la stragrande maggioranza delle applicazioni aziendali). Occorre infatti mantenere in sincrono la base dati con le modifiche del codice, variando la prima in accordo ai cambiamenti del secondo. La modifica dei dati proposta dall'Agile Data è chiamata *database refactoring*. Un database refactoring è una semplice modifica al database, in modo che questi ne risulti migliorato, ma senza modificare gli aspetti comportamentali e semantici. Il database refactoring è potenzialmente più complesso del refactoring del codice, in quanto un database è composto sia da dati (le tabelle), che programmi (stored procedures). Inoltre, il codice deve mantenere solo gli aspetti comportamentali, mentre il database deve mantenere anche quelli *semantici*. Per semantica dei dati, si intende il significato delle informazioni contenute nel database dal punto di vista dell'utilizzatore. Un campo *priorità*, ad esempio, potrebbe contenere i valori 1 (bassa), 2 (media), 3 (alta): l'utilizzo dei valori numerici con questo significato è la semantica del campo.

Il refactoring del database è difficile perché esiste un altro livello di accoppiamento tra il database ed i programmi: nel caso più semplice, la base dati è utilizzata solo dalla singola applicazione in esame. Ma in casi più complessi, possono esistere altre applicazioni che accedono direttamente a quei dati, oppure script di importazione ed esportazione dati, codice di test, framework di persistenza, altri database collegati, oppure altre applicazioni di cui si è a conoscenza – o altre di cui non si sa nulla.

Agile Data propone un catalogo di refactoring ed un processo di modifica. Il processo, in breve, è composto da una serie di passaggi:

- verifica che il refactoring sia effettivamente necessario;
- scelta di quello più opportuno;
- determinazione delle pulizie di dati richieste;
- scrittura dei test di unità;
- deprecazione dello schema originale;
- implementazione delle modifiche;
- aggiornamento degli script di migrazione;
- esecuzione dei test di regressione;
- documentazione del refactoring;
- controllo di versione.

Le pratiche AD prevedono l'uso di una *sandbox*, un ambiente di sviluppo che isoli le modifiche in corso al database e di dati di test, indispensabili per eseguire i test.
Il processo AD viene utilizzato in congiunzione al catalogo di refactoring che, in modo similare a quello di Fowler, descrive motivazioni e tecniche per migliorare la struttura del database esistente.

Feature Driven Programming

La programmazione orientata alle funzionalità si basa su una scaletta composta da cinque passi:

sviluppo di un modello complessivo. Impegna il 10% del tempo all'inizio ed il 4% durante il corso del progetto. Se ne occupa il team di "modellatori", esperti verticali della materia da trattare, che si preoccupa di creare un modello complessivo del dominio del problema. Il modello è aggiornato in modo iterativo in funzione dei risultati emersi dal quarto passo;

costruzione di una lista di funzionalità. Sviluppato per il 4% all'inizio e per l'1% nel corso del progetto, comporta la suddivisione del modello complessivo in aree più grandi. Queste vengono poi spezzate in singole attività, che prendono il nome di funzionalità (features). Il risultato è organizzato in una lista con struttura gerarchica. Si noti che nella FDD una feature può essere anche tecnica: non c'è dunque una diretta relazione tra requisito funzionale e feature;

pianificazione sulla base delle funzionalità. La pianificazione occupa il 2% del tempo iniziale ed il 2% del tempo nel corso del progetto del team dedicato alla

pianificazione. Questo include sia il capo progetto, che il responsabile di sviluppo che i capi programmatori, figure che si occupano di coordinare i singoli sviluppatori. La pianificazione avviene con le consuete modalità, considerando le attività da svolgere e le relative priorità;

progettazione delle funzionalità. Occupa, insieme al passo seguente di costruzione, il 77% del tempo delle risorse. Lo sviluppo è coordinato dai capi programmatori, che hanno la responsabilità di identificare i proprietari delle singole classi (i programmatori) e riunirli per fargli sviluppare una determinata funzionalità. In questo aspetto, FDD è in aperto contrasto con Extreme Programming, dove la proprietà del codice è collettiva;

costruzione delle funzionalità. Nella fase di implementazione il codice viene testato con i test di unità, ma anche revisionato dai capi programmatore. Quando il codice passa queste due verifiche, viene promosso a codice di rilascio: viene cioè inserito nel repository del codice di progetto condiviso da tutti gli sviluppatori. Lo scopo di questa fase è quello di soddisfare uno più requisiti funzionali – elementi che hanno "valore" per il cliente (client-valued feature).

Adaptive Software Development di Highsmith

A testimonianza del fatto che le metodologie agili sono il risultato di un processo evolutivo nato in ambienti diversi, si considernino le metodologie di Highsmith: queste sono il risultato di anni di esperienza dell'autore con metodologie predittive. Dopo averle sviluppate, utilizzate ed insegnate, questi è giunto alla conclusione che queste hanno problemi fondamentali, almeno per il mondo degli affari odierno.
Il risultato dello studio di Highsmith è ASD (Adaptive Software Development), una metodologia adattiva basata su tre fasi non lineari: la speculazione, la collaborazione e l'apprendimento. Quest'ultima fase è interessante ed in realtà intende individuare il processo di sviluppo: nelle discipline predittive, l'apprendimento è scoraggiato: il progetto viene creato all'inizio e deve essere seguito nelle modalità predefinite; in ASD il risultato dello sviluppo di ogni micro-fase è utilizzato per imparare come meglio affrontare la fase successiva.
Il libro relativo ad ASD non fornisce pratiche specifiche come esistono nell'Extreme Programming, ma fornisce una serie di indicazioni di base sul perché lo sviluppo adattivo è importante e le consegenze che l'adozione di questo ha a livello organizzativo e del management. Due elementi interessanti di ASD è che le idee di base derivano dalla teoria del caos (sistemi adattivi complessi) e che le deviazioni, nelle discipline

tradizionali, sono viste come errori da correggere, mentre nelle discipline adattive queste sono vieste come guide per raggiungere la soluzione corretta.

Considerazioni sull'applicabilità e problemi delle metodologie agili

Alcuni aspetti delle tecniche agili di sviluppo come l'Extreme Programming non sono facilmente applicabili per vari motivi:

vaghezza. Uno dei problemi dell'Extreme Programming è la vaghezza in merito alla concreta applicazione delle pratiche: le iterazioni di produzione quanto devono essere lunghe? Nel creare *test di unità*, cosa si intende per unità? Fowler stesso [web3], non sa dare una risposta precisa, rimandando la determinazione se l'unità è una classe od un sottosistema all'esperienza, e consigliando, per rispondere a questa domanda, di cominciare a svilupparne, per capirlo in prima persona;

funziona solo per buoni programmatori. Una delle critiche mosse contro XP è che questa funziona solo per buoni sviluppatori, ad esempio, come Kent Beck che hanno l'esperienza per progettare bene, semplicemente ed in modo estensibile fin da subito, mentre altre persone, meno preparate, avrebbero delle difficoltà ad ottenere lo stesso risultato. Anche il concetto di proprietà collettiva è un aspetto

che funziona solo con buoni programmatori: quelli meno esperti potrebbero tendere a non capire il sistema nella sua interezza o non sentire sufficientemente la responsabilità del corretto funzionamento del sistema;

pair programming. Alcune volte può semplicemente capitare che non ci siano sufficienti sviluppatori in un gruppo di lavoro per supportare il pair programming. Il numero minimo di elementi del team necessario per applicare questa tecnica è ovviamente di due risorse, ma anche con un gruppo di lavoro di quattro persone, un numero medio per sviluppi modesti, il numero di attività conducibili in parallelo è di solo due attività. Questa è banale matematica, ma con il pair programming, il numero di sviluppatori allocabili sulle attività esaurisce presto e, sebbene la promessa dello sviluppo agile sia quella di ottenere in breve tempo risultati tangibili, questa pratica può dare l'impressione al managment di progetto di una lenta progressione dello sviluppo. In realtà, l'apporto di due sviluppatori su un singolo problema, soprattutto se questo è di difficile soluzione, può essere una pratica utile. In effetti, è una pratica di buon senso, che però l'extreme programming porta, appunto, all'estremo, suggerendo la programmazione di coppia, *per tutto il tempo dello sviluppo*.
Un altro problema è legato al livello di attenzione che può porre il secondo sviluppatore: se inesperto può non mantenere

la concentrazione. Il pair programming non è l'affiancamento per la formazione: le due risorse possono essere di qualsiasi livello, ma devono avere preparazione simile.
Esistono però anche esempi di successo del pair programming, con addirittura la realizzazione di un triple programming [5].
Si noti, inoltre, che il pair programming non è una pratica volta al raggiungimento veloce dell'obbiettivo, ma alla qualità del codice;

unit testing. Se per un progetto in fase di definizione lo sviluppo dei test di unità può essere una pratica perseguibile, è più difficile eseguire un refactoring di un progetto esistente, soprattutto se questo contiene un numero considerevole di classi ed il tempo a disposizione è poco. Inoltre, il tempo che richiederebbe la creazione dei test è tale da essere in conflitto con i tempi iterativi ristretti richiesti dalle metodologie agili.
Si noti comunque, che la creazione di unit test comporta sostanzialmente il raddoppiamento del lavoro, in quanto deve essere scritto il codice effettivo ed il codice di test. Inoltre, per *unit* si intende, soprattutto per chi è alle prime armi con questa pratica, la singola classe. Questo comporta la presenza di una classe di test per ciascuna classe dell'applicazione, cosa che comporta una decisa proliferazione di codice.

La presenza del doppio delle classi comporta anche problemi in fase di manutenzione, in quanto anche questo è codice che deve essere gestito, modificato e documentato. In quest'ottica si evidenzia l'indispensabilità di tenere sempre in sincrono le classi del programma con gli *unit test* e con le specifiche applicative;

orientamento alle persone. Il problema del morale degli sviluppatori è un classico aspetto che il management di un progetto software deve affrontare. Sebbene i metodi agili siano orientati alle persone, questo aspetto è di difficile attuazione, anche per il fatto che *esce dall'ambito tecnico per entrare in quello manageriale*. La sua applicazione prevede dunque che il manager sposi la metodologia. Questo aspetto ha un impatto diverso rispetto alle altre pratiche agili perché il manager passa da ruolo passivo (osservare la bontà delle pratiche), a quello attivo (applicare una pratica in prima persona).

costante disponibilità del cliente. Se da una parte l'Extreme Programming richiede la costante presenza del cliente per potergli sottoporre domande chiarificatrici sui requisiti utente, dall'altra è bene considerare il tipo di collaborazione che ci si può aspettare dal cliente. Spesso questi non collabora ed anzi ha interesse nel mantenere i

requisiti il più torbido possibili ed eventualmente a non arrivare mai alla fine del progetto. Quando infatti l'ambito si sposta a livello politico/gestionale, hanno più peso gli intrecci e gli interessi politici rispetto alla soddisfazione degli utenti ed agli aspetti tecnici. In questi casi il fatto di congelare i requisiti è un aspetto fondamentale, per evitare che lo sviluppo vada avanti all'infinito, anche in virtù del fatto che i contratti sono fatti sulla base del raggiungimento di un obiettivo e non a tempo (come anche sostiene Fowler, [3]). Questi clienti, infatti, si guardano bene dall'instaurare contratti di consulenza a giornate: sarebbero esposti in prima persona alla loro inefficienza: prediligono i contratti a progetto, dove possono cambiare idea in qualsiasi momento senza che questo comporti aumenti di costo. In questo caso il congelamento dei requisiti è una minima ancora di salvezza per il fornitore che può così cercare di arginare le fantasie del cliente, ma è anche un aspetto che entra in conflitto con tutta la logica delle metodologie agili, di fatto promuovendo un aumento delle dimensioni delle iterazioni di sviluppo. In altre parole, anche suddividendo un progetto in fasi la cui analisi viene sviluppata in modo sequenziale, si hanno grossi blocchi di specifiche congelate su cui può essere più efficiente lo sviluppo tradizionale al posto di quello agile, per il fatto che non è richiesta flessibilità all'interno di quel blocco di sviluppo. In questo caso i benefici di aver sviluppato secondo pratiche agili

si avrebbe potenzialmente solo tra un blocco e l'altro, aumentando la richiesta di sforzo iniziale.

Prodotti per il refactoring

Alcune pratiche agili possono essere di noiosa attuazione e questo può facilmente portare ad errori. Il refactoring, in particolare, necessita di strumenti appositi per poter essere applicato in sicurezza e velocità.

JRefactory

E' un software open source che dispone di un catalogo di 15 refactoring e l'integrazione con IDE quali JBuilder di Borland, Netbeans (parziale), Elixir. Dispone inoltre di strumenti per lo sviluppo, quali semplici strumenti per la metrica del codice.
> http://jrefactory.sourceforge.net/csrefactory.html

JFactor

Questo prodotto commerciale di Instantiations offre un catalogo di refactoring di codice studiato congiuntamente a Martin Fowler. Si trovano refactoring specifici per il linguaggio Java; il prodotto si integra con ambienti IDE come Visual Age di IBM e JBuilder di Borland.
> http://www.instantiations.com/jfactor/default.htm

RefactorIT

Questo strumento di refactoring ha integrazione con IDE come JDeveloper di Oracle, SunONE Studio di SUN e JBuilder di Borland. Dispone inoltre di metriche e ricerche sul codice.

> http://www.refactorit.com/index.html

Eclipse

Questo ambiente di sviluppo è stato creato da IBM ed ora è manutenuto dal consorzio omonimo. L'ambiente contiene alcuni refactoring direttamente integrati.

> http://www.eclipse.org

Risorse Web italiane

Italian Agile Movement

Portale italiano di notizie, risorse e documentazione sulle discipline agili.

> http://www.agilemovement.it/

Hacking Extreme Programming

Uno spazio web non commerciale dedicato all'introduzione e la divulgazione delle metodologie XP ed in generale dei processi di sviluppo software "lightweight".

> http://www.hxp.it/

Risorse Web straniere

Manifesto for Agile Software Development

Sito che raccoglie gli elementi comuni a tutte le discipline agili.

> http://agilemanifesto.org

Agile Alliance

Consorzio di organismi ed aziende che condividono la visione "agile" dell'approccio allo sviluppo del software.

> http://www.agilealliance.org/

Feature Driven Development

Portale di Jeff DeLuca sulla FDD - Feature Driven Development

> http://www.featuredrivendevelopment.com/

Scrum Development Process

Il sito ufficiale della metodologia SCRUM.

> http://www.controlchaos.com/

Extreme Programming: A Gentle Introduction

Il sito dedicato all'Extreme Programming: documentazione, pratiche, risorse ed eventi.

> http://www.extremeprogramming.org

XProgramming

Portale dedicato all'Extreme Programming. Ospita un magazine con articoli su XP, una community e recensioni di libri.

> http://www.xprogramming.com/

Agile Modeling

Sito dedicato alla modellazione agile. Presenta valori, concetti e pratiche della modellazione agile. Sono presenti phamplet liberamente scaricabili.

> http://www.agilemodeling.com

Agile Data

I principi dello sviluppo agile del software portati ai database. Il sito ha una completa documentazione sui database relazionali, il mapping ad oggetti, processi di modifica ed un catalogo di refactoring orientati ai dati.

> http://www.agiledata.org/

Bibliografia

[1] César et.alt , *Un nuovo metodo di sviluppo software: eXtreme Programming (XP)*, http://apache.tecnoteca.it/upgradepdf/it-up3-2Acebal.pdf

[2] G.Bellavia , *Introduzione alla Extreme Programming*, http://www-lia.deis.unibo.it/Courses/IngSwA0102/Relazioni/ExtrProg

[3] Martin Fowler , *The New Methodology*, http://www.martinfowler.com/articles/newMethodology.html

[4] Daniel H.Steinberg *, Does anyone really XP? – Making XP scale to large projects and an integrated refactoring tool*, Giugno 2001. http://www-106.ibm.com/developerworks/java/library/j-j1xp.html

[5] Sanjay Murthi, *Project Management and Agile Methods*, Giugno 2002, http://www.sdtimes.com/opinions/guestview_056.htm

Riferimenti

[web1] Extreme Programming: A Gentle Introduction, http://www.extremeprogramming.org

[web2] Agile Modeling, http://www.agilemodeling.com

[web3] Test-Driven Development – A conversation with Martin Fowler, Part V, http://www.artima.com/intv/testdriven2.html

[web4] Sito dell'Agile Alliance, http://agilealliance.org

[web5] Manifesto del movimento agile, http://agilemanifesto.org

Libri

[libro1] Martin Fowler, *Refactoring: Improving the Design of Existing Code*, Addison Wesley, 1999. ISBN 0201-48567-2

[libro2] Kent Beck, *Extreme Programming Explained: Embrace Change*, Addison Wesley Longman, 2000. ISBN 201-61641-6

[libro3] Peter Coad et al, *Java Modeling in Color with UML*, Prentice Hall PTR, 1999. ISBN 013-01151-0X

L'autore

Massimiliano Bigatti é autore del libro "Da Visual Basic a Java". E' certificato, tra le altre, come *SUN Certified Enterprise Architect for Java Platform, Enterprise Edition Technology*. Si occupa di architetture applicative basate su Java, technical writing e del portale http://javawebservices.it.

Java Web Services.it

Java Web Services.it, il portale italiano dedicato ai Web Services in Java.