Sandro Pedrazzini

Tecniche di progettazione agile con Java: design pattern, refactoring, test

Sommario

Capitolo 12. Applicare i pattern

Prefazione

"Bad programming languages and young programmers without experience were the main cause of the dot-com crashes […]. You need much time to become a good software engineer, you cannot be a good developer after only 3-5 years of study […]"

(Cattivi linguaggi di programmazione e giovani programmatori senza esperienza sono stati la causa principale dei fallimenti nelle aziende della *new economy*... Occorre un certo periodo di tempo per diventare un buon ingegnere del software: non si diventa sviluppatori bravi in soli 3-5 anni di studio...)

RICHARD P. GABRIEL

Tecniche di progettazione agile. Non sapevo in che altro modo caratterizzare questo libro, fino a quando l'ho ripreso, capitolo per capitolo, per un'ultima completa rilettura, prima di consegnarlo egli editori. Mi sono allora accorto che c'era un elemento ricorrente in sottofondo, una sorta di filo conduttore, che pur nascosto, riusciva a valorizzare certi argomenti: l'esperienza.

Esperienza in informatica. Quando mai? Sono così andato a riprendere vecchi appunti di una presentazione a una conferenza a cui ho partecipato nel 2002 a San Francisco (International Lisp Conference: ebbene sì, anche se da anni insegno Java e la mia azienda sviluppa prevalentemente in Java, la mia vera passione è un'altra…). Ho ritrovato le annotazioni scritte durante la presentazione di Richard Gabriel, *invited speaker*, di Sun Microsystems, nonché professore a Stanford, noto sia per i suoi lavori e le sue pubblicazioni, sia per le sue posizioni critiche nei confronti dei metodi di insegnamento dell'informatica nelle università.

In quell'occasione Gabriel affermò che uno dei motivi principali dei problemi avuti durante gli anni 2000-2002 dalle cosiddette *dot-com* (cioè le aziende che fondano il loro business su Internet) e che portarono poi al ridimensionamento della new economy fu senza dubbio la mancanza di esperienza. A programmatori giovani, appena laureati o ancora studenti, venivano date incombenze e responsabilità troppo elevate. In teoria conoscevano perfettamente le tecnologie, ma non avevano alle spalle esperienza sul campo che potesse aiutarli nella gestione dei progetti e, soprattutto, nella loro manutenzione ed evoluzione.

Ecco cosa c'entra l'esperienza. Ecco perché non è un paradosso parlare di esperienza in informatica. La mancanza di esperienza ha fatto fallire molti progetti, ha fatto gettare parecchi soldi dalla finestra, ha fatto dubitare più di una persona della reale capacità degli informatici di gestire progetti dall'inizio alla fine, portandoli al successo.

Perché parlare di esperienza in un libro che tratta di progettazione agile? Perché per un buon design è necessario avere esperienza: del resto i design pattern nascono dall'esperienza e sono un modo per sfruttare quella acquisita in progetti passati, addirittura in progetti di altri. L'esperienza è ciò che ci porta a dire che un programma, con la sua architettura, è in continua evoluzione. È con l'esperienza che dobbiamo gestire questa evoluzione, applicando il refactoring e rivedendo in modo continuo il design.

Non ci sono magie. Non possiamo fare a meno dell'intervento dello sviluppatore. Nelle metodologie agili, la programmazione non è un dettaglio del progetto, ma è la parte essenziale, che comprende design, refactoring e test. La programmazione non è considerata l'ultimo gradino del progetto, perché i moderni linguaggi di programmazione permettono la creazione di design partendo dal codice; perché gli ultimi ambienti di sviluppo sono sofisticati a tal punto da facilitare refactoring e redesign. La capacità di programmazione è quindi essenziale e nella programmazione è richiesta esperienza. Nelle metodologie agili l'esperienza è un fattore determinante.

Parlare di utilizzo pratico di tecniche di progettazione agile significa considerare negli aspetti di progettazione l'esperienza acquisita nello sviluppo. Significa fondere, non separare, la fase di progettazione e quella di realizzazione. Nelle metodologie agili il concetto di esperienza ricorre sotto forma di capacità di gestire il software in modo adattivo. I cicli di sviluppo sono corti e la separazione tra progettazione e sviluppo è minima. Un principio fondamentale nei metodi agili consiste nel fatto che i processi e gli utensili a disposizione non hanno la pretesa di sostituire in nessun caso il team di sviluppo. Servono invece da supporto al lavoro di design e implementazione, che rimane completamente a carico dello sviluppatore.

L'idea di adattività significa anche relativizzare l'importanza del primo design. Va studiato, certo, va discusso, ma modifiche nei requisiti durante la fase di realizzazione non vanno viste come attacchi al progetto, bensì normali elementi di realizzazione, che in ultima analisi fanno crescere ed evolvere il design.

Gabriel concluse dicendo che uno studente, dopo 3-5 anni di studio, non è pronto per ogni tipo di progetto. Dovrà formarsi sul campo. Questa non è certo una novità, vale in tutti i settori, ma in informatica si tende a dimenticarlo, accecati dal fatto che i nuovi laureati sembrano conoscere meglio di altri le ultime tecnologie. La tecnologia non è però tutto: è una sola tra le componenti necessarie affinché un progetto vada a buon fine.

Nella mia doppia veste di docente, con contatti settimanali con studenti di informatica, e di ingegnere software, alle prese giornalmente con progetti di sviluppo, non posso che condivide-

re le paure di Gabriel. Ci dobbiamo quindi chiedere come valorizzare l'esperienza in un mondo di tecnologie che si evolvono e cambiano a ritmi così frenetici; come far fruttare il bagaglio di conoscenze acquisite in progetti passati. La capacità di individuare il corretto design, il coraggio di farlo evolvere, la diligenza nello scrivere test che rendano robusto il codice nei confronti delle modifiche, sono gli elementi essenziali della progettazione agile, che rimarranno costanti per parecchio tempo e sui quali vale la pena investire e costruirsi esperienza.

In questo libro, nato come corso universitario, ma frutto anche dell'esperienza come ingegnere software presso Canoo, azienda di sviluppo e consulenza attiva in grossi progetti IT, cerco di mostrare quali sono a mio giudizio gli elementi su cui uno studente deve fondare la propria conoscenza di progettazione. Sono, entro certi limiti, indipendenti dalla tecnologia usata, ma sono elementi pratici, che vanno imparati e poi ripetutamente esercitati e impiegati nel lavoro quotidiano di progettazione e sviluppo.

Il taglio del libro è molto pratico, essendo questi elementi introdotti in un corso di bachelor con sbocco professionale (il polo universitario in cui tengo questo corso prepara gli studenti al master, ma ha anche lo scopo di formare ingegneri in grado di entrare nel mondo del lavoro, se lo desiderano, dopo i primi tre anni).

Struttura del libro

Il libro, dopo un capitolo introduttivo, si sviluppa in quattro parti, ognuna delle quali è formata da capitoli distinti, in modo da permettere una lettura anche frammentata delle singole sezioni. Gli esempi sono tutti implementati in Java, linguaggio di riferimento per l'intero libro.

I La prima parte si prefigge di dare una visione generale dei concetti attraverso un esempio iniziale. Viene sviluppata una prima soluzione, ripresa e poi modificata in modo iterativo sulla base di nuovi requisiti ed esigenze. Gli aspetti trattati sono quelli del design object-oriented in generale e degli aspetti approfonditi nelle parti a seguire, cioè design pattern, refactoring e test.

II I design pattern vengono trattati nella seconda parte, che prevede, oltre a un'introduzione sintetica sui concetti della programmazione a oggetti, capitoli riguardanti le diverse definizioni di pattern, diversi esempi di pattern, una modifica passo per passo di un'applicazione e accenni su nuovi pattern.

III La terza parte prevede la spiegazione del concetto di test come elemento di sviluppo, facendo riferimento al framework di test JUnit, largamente diffuso e utilizzato per la programmazione in Java.

IV I metodi di refactoring vengono trattati nella quarta parte, dove, di nuovo, oltre alla spiegazione dei concetti più significativi e all'approfondimento di alcuni metodi, l'elemento più importante è costituito dallo sviluppo di un paio di esempi e dalle loro relative modifiche.

Citazioni

Essendo il libro nato come corso, mi sono riferito a parecchie pubblicazioni, libri e materiale disponibile in rete. Non mancherò di citare le fonti di alcune idee o esempi usati, anche se, per la maggior parte dei casi, gli adattamenti effettuati per integrarli nei vari temi dei capitoli, li hanno allontanati parecchio dalla loro versione originale (sarei naturalmente ben felice se a loro volta, alcuni esempi di questo libro venissero usati per altre pubblicazioni o corsi). Vorrei comunque già citare le fonti più importanti, quelle che hanno dato lo stimolo iniziale a questo lavoro, anche perché aiutano a dimostrare che ci sono elementi, anche nel campo della progettazione e dello sviluppo, che durano nel tempo.

Il primo libro è senza dubbio quello della cosiddetta *gang of four* sui design pattern ([Gamma-1995]), pubblicato nel 1995 e tuttora più che attuale. Il secondo è invece il libro di Martin Fowler sui refactoring ([Fowler-1999]), dal quale ho preso spunto per alcuni esempi pratici riportati nel testo. A questi vanno aggiunti alcuni libri di Kent Beck sulla programmazione estrema, la cosiddetta XP, e l'utilizzo del test in modo sistematico, pubblicati nella seconda metà degli anni Novanta, sfociati, per quanto riguarda questo testo, nel libro sullo sviluppo a partire dal test ([Beck-2003]), pubblicato nel 2003.

Appunti personali

All'interno dei singoli capitoli, mi sono permesso di inserire alcune note personali. Si tratta a volte di approfondimenti, a volte di aneddoti raccolti nel corso della mia esperienza professionale. Non sono indispensabili per la comprensione del testo, possono essere tranquillamente ignorati, sono però senz'altro utili ad alleggerirne il contenuto.

Ringraziamenti

Mai mi sarei immaginato che la strada per passare da un testo di corso, già ben fornito di esempi e spiegazioni, a un libro, fosse così lunga. Sono molte le persone a cui devo parecchio in questa mia avventura editoriale.

In primo luogo a Giambattista Ravano, collega e direttore del dipartimento in cui insegno, che mi ha dato la possibilità di portare questi temi nel corso di progettazione software. Durante le lezioni dell'anno accademico 2003/2004 sono stato assistito da Alessandro Trivilini, che mi ha sgravato di alcuni compiti e mi ha fornito il necessario feedback. A lui, ai colleghi dell'Istituto di Ingegneria del Software e Sistemi Informativi (ISSI) e alle classi del terzo anno, che si sono date la pena di leggere il testo e inviarmi i commenti, vanno i miei ringraziamenti.

Non avrei mai potuto proporre un corso del genere senza l'indispensabile esperienza dei miei colleghi di Canoo, pionieri nell'introdurre metodologie agili nella progettazione e nello sviluppo. Essendo attivi in parecchi progetti, abbiamo l'occasione di confrontarci regolarmente e discutere soluzioni e metodologie. Discussioni a questo riguardo ne ho avute anche con i colleghi di Tinext, ditta con cui ho regolare occasione di collaborare, che ringrazio per la disponibilità.

Un ringraziamento speciale va inoltre a Giovanni Puliti, direttore della rivista internet MokaByte, che ha subito creduto in questo progetto e mi ha dato il necessario supporto, assieme a Francesco Saliola, a cui devo il prezioso aiuto redazionale.

Da ultimo, mi si permetta un ringraziamento speciale alla mia famiglia. Dedico questo libro in particolar modo alle piccole Sabina e Sofia, che per parecchi mesi hanno avuto, di sera e durante i fine settimana (i momenti dedicati al libro, assieme alle vacanze in montagna e ai viaggi in treno per e da Basilea), un papà presente fisicamente, ma poco incline ai giochi rumorosi; e a mia moglie Francesca, abituatasi, per amore o per forza, durante le fasi finali del libro, a ripetermi le richieste almeno tre volte prima di osservare una mia reazione (solitamente, dice, ne bastano due). Grazie anche a loro.