



SANDRO PEDRAZZINI

# Elementi di progettazione agile: utilizzo pratico di design pattern, refactoring e test





# Sommario

## Prefazione

Struttura del libro .....	iii
Citazioni .....	iv
Appunti personali .....	iv
Ringraziamenti .....	iv

## Capitolo 1. Introduzione

Metodi agili .....	2
Imprevedibilità .....	5
Design pattern .....	5
Metodi di refactoring .....	6
Test di unità .....	6
Obiettivi della progettazione .....	6
Affidabilità .....	6
Modificabilità .....	6
Comprensione .....	7
Riutilizzo .....	7
Conseguenze .....	7

## Parte I Visione generale: verso un design object-oriented

### Capitolo 2. Problema iniziale

Descrizione .....	12
Primo design .....	12
Funzionalità .....	16
Test .....	17
Programma principale di prova .....	18
In questo capitolo .....	19

**Capitolo 3. Prima verifica e fattorizzazione**

Delega di funzionalità .....	22
Test .....	26
Ulteriore delega di funzionalità .....	26
In questo capitolo.....	28

**Capitolo 4. Fattorizzare per il test**

Estrazione dei metodi .....	29
Problemi di performance .....	31
In questo capitolo.....	33

**Capitolo 5. Problemi di classificazione**

Ulteriore delega di responsabilità .....	35
Eliminare lo switch .....	38
In questo capitolo.....	42

**Capitolo 6. Condivisione della sequenza**

Trovare gli elementi ridondanti .....	45
In questo capitolo.....	49

**Parte II Design pattern****Capitolo 7. Introduzione all'idea di pattern**

"Il" libro .....	53
Visione del software .....	54
Flessibilità .....	55
Definizione .....	56
Documentazione .....	56
In questo capitolo.....	56

**Capitolo 8. Design e programmazione object-oriented**

Information hiding e data abstraction .....	58
Ereditarietà .....	59
Motivi per l'ereditarietà .....	61
Composizione .....	61
Visibilità .....	63
Polimorfismo e binding dinamico .....	63
Polimorfismo .....	63
Binding .....	64

Verifica dei tipi .....	65
Tipi e classi .....	66
Concetto di riutilizzo .....	66
Adattabilità .....	66
Localizzazione .....	66
Astrazione .....	66
In questo capitolo .....	68

## Capitolo 9. Definizioni di design pattern

Definizioni .....	69
Descrizione nel catalogo .....	70
Un esempio di catalogo: il pattern Proxy .....	71
Scopo .....	71
Altro nome .....	71
Motivazione .....	71
Applicabilità .....	72
Esempi .....	72
Struttura .....	72
Partecipanti .....	72
Collaborazioni .....	72
Conseguenze .....	72
Esempi di codice .....	73
Utilizzi noti di Proxy .....	78
Pattern simili .....	78
In questo capitolo .....	78

## Capitolo 10. Utilizzo di un pattern: Iterator

Situazione iniziale .....	79
Prime modifiche .....	80
Discussione .....	80
Secondo passaggio .....	81
Terzo passaggio: utilizzo polimorfico .....	82
Quarto passaggio: metodo factory .....	83
In questo capitolo .....	84

## Capitolo 11. Affrontare il design

Terminologia dei requisiti .....	85
Responsabilità e collaborazioni .....	86
Diagrammi di sequenza .....	87
Design evolutivo .....	87
Ruolo dei pattern .....	87
Primo principio .....	88

Program to an interface, not to an implementation .....	88
<b>Ereditarietà e composizione .....</b>	<b>88</b>
Ereditarietà .....	89
Composizione .....	89
Secondo principio .....	90
Favor object composition over class inheritance .....	90
<b>Delegation .....</b>	<b>90</b>
<b>Design for change .....</b>	<b>91</b>
<b>Cause di redesign .....</b>	<b>91</b>
Creazione di oggetti attraverso il nome di una classe .....	91
Dipendenza da operazioni specifiche .....	91
Dipendenza da piattaforme software o hardware .....	92
Dipendenza da rappresentazioni o implementazioni di oggetti .....	92
Dipendenza algoritmica .....	92
Associazioni troppo strette .....	92
Estensione di funzionalità attraverso la gerarchia .....	92
Impossibilità di modificare classi in modo conveniente .....	92
<b>Livelli di riutilizzo .....</b>	<b>93</b>
Applicazione .....	93
Biblioteche di classi .....	93
Framework .....	93
<b>In questo capitolo .....</b>	<b>95</b>

## Capitolo 12. Applicare i pattern

Specifiche iniziali .....	97
<b>Nuove richieste .....</b>	<b>98</b>
Ordine e stream .....	98
Trasformazione del contenuto .....	98
Informazioni supplementari sul contenuto .....	98
<b>Realizzazione diretta .....</b>	<b>99</b>
<b>Nuovo design .....</b>	<b>101</b>
<b>Trasformazioni .....</b>	<b>104</b>
Strategy .....	105
Utilizzo .....	105
Conseguenze .....	105
<b>Sequenza di output .....</b>	<b>109</b>
Template .....	109
Utilizzo .....	109
Conseguenze .....	109
<b>Informazioni supplementari .....</b>	<b>111</b>
Decorator .....	112
Utilizzo .....	112
Conseguenze .....	112
<b>Diversi canali di output .....</b>	<b>116</b>
Observer .....	116
Utilizzo .....	116

Conseguenze .....	116
Polimorfismo .....	118
In questo capitolo .....	120

## Capitolo 13. Pattern per il Web

Architetture Web .....	121
Architettura MVC .....	121
Controller per architetture Web .....	122
Pattern Transform View .....	123
Esempio .....	124
Applicazione Web .....	126
Pattern Two Step View .....	126
Descrizione dei passaggi .....	126
Vantaggi .....	127
Svantaggi .....	128
In questo capitolo .....	128

## Capitolo 14. Canoo ULC e il pattern Half-Object

Limiti delle applicazioni Web .....	129
Canoo ULC .....	130
Utilizzo di ULC .....	132
Confronto con Swing .....	132
Aggiunta di eventi .....	133
Il pattern Half-Object .....	134
Problema .....	134
Motivazione .....	134
Implementazione .....	134
Esempi di utilizzo .....	135
Esempio con codice .....	135
Architettura client/server .....	136
Sincronizzazione .....	136
Protocollo .....	136
Client .....	137
Server .....	138
In questo capitolo .....	141

## Capitolo 15. A caccia di pattern

Gestione degli eventi .....	143
Documentazione con pattern .....	144
Observer .....	146
Subject .....	147
Realizzazione in Java .....	147
Gestione del layout .....	147
Component e Container .....	149

Ulteriori pattern .....	150
Nuovi pattern .....	151
In questo capitolo.....	152

## Parte III Test di unità

### Capitolo 16. Test automatico

Necessità di test .....	155
Fretta e termini di consegna.....	155
Eccessiva sicurezza .....	156
Formazione sbagliata .....	156
Poca professionalità .....	156
Benefici .....	156
Come scrivere test.....	157
Come facilitare il test .....	158
Metodi di visualizzazione .....	158
Ogni classe eseguibile .....	158
Classi di supporto .....	159
Utilizzo di un tool esterno .....	160
In questo capitolo.....	160

### Capitolo 17. JUnit

Utilizzo di assert .....	162
Hello World .....	164
Chiamate .....	165
La classe Assert .....	166
Inizializzazioni comuni .....	167
Ridurre il test all'essenziale .....	169
In questo capitolo.....	169

### Capitolo 18.

#### Sviluppare con JUnit

Descrizione del problema .....	171
Caso semplice .....	172
Prima codifica .....	172
Test .....	173
Inizializzazioni comuni .....	174
Esecuzione .....	174
Utilizzo di più valute .....	175
Esempio .....	175
Codifica .....	175
Test .....	176



Nuovo add() .....	177
In questo capitolo .....	178

## Capitolo 19. Design by testing

Adattamento del design .....	179
Test .....	180
Realizzazione .....	181
Dispatching semplice .....	182
Dispatching doppio .....	183
Soluzione: dispatching manuale .....	185
Resto del codice .....	187
Semplificazioni .....	188
In questo capitolo .....	189

## Capitolo 20. Test-driven development

La teoria .....	192
La pratica .....	193
Problema .....	193
Procedimento .....	193
Prima strategia: implementazione "fasulla" .....	194
Nuovi passi .....	195
Effetti collaterali .....	195
Seconda strategia: realizzazione diretta .....	196
Controllo di uguaglianza .....	196
Terza strategia: triangolazione .....	196
Valute diverse .....	197
Generalizzazione .....	198
In questo capitolo .....	200

## Parte IV Refactoring

### Capitolo 21. Introduzione al refactoring

Impatto sul design .....	204
Utilizzo sistematico durante lo sviluppo .....	204
Miglioramento del design .....	204
Software più leggibile .....	205
Si scoprono errori in anticipo .....	205
Si programma più velocemente .....	205
Passaggi di refactoring .....	206
Esempio .....	207
Elenco di alcuni refactoring .....	208
Refactoring Extract .....	208
Extract Class .....	208

Extract Hierarchy .....	208
Extract Interface .....	208
Extract Method .....	208
Extract Subclass .....	208
Extract Superclass .....	209
<b>Refactoring Push/Pull .....</b>	<b>209</b>
Pull Up Constructor Body .....	209
Pull Up Field .....	209
Pull Up Method .....	209
Push Down Field .....	209
Push Down Method .....	209
<b>Refactoring complessi .....</b>	<b>209</b>
Replace Conditional with Polymorphism .....	209
Replace Inheritance with Delegation .....	209
Replace Delegation with Inheritance .....	210
Separate Domain from Presentation .....	210
<b>In questo capitolo...</b>	<b>210</b>

## Capitolo 22. Primi esempi di refactoring

<b>Aggiunta di un metodo factory .....</b>	<b>211</b>
Scopo .....	211
Motivazione .....	212
Schema di trasformazione .....	212
Procedimento .....	212
Esempio .....	213
Vantaggi .....	214
Reflection .....	215
<b>Modifica della relazione tra classi .....</b>	<b>215</b>
Scopo .....	215
Motivazione .....	215
Procedimento .....	217
Esempio .....	217
<b>In questo capitolo...</b>	<b>219</b>

## Capitolo 23. Refactoring complessi

<b>Modello MVC .....</b>	<b>221</b>
Pattern Observer .....	222
<b>Esempio .....</b>	<b>223</b>
Passaggi .....	224
Descrizione .....	224
<b>Primo passaggio di trasformazione .....</b>	<b>226</b>
<b>Secondo passaggio di trasformazione .....</b>	<b>228</b>
Creazione dei metodi .....	228

Sostituzione dell'accesso diretto .....	229
Aggiornamento con input dell'utente .....	229
Terzo passaggio di trasformazione .....	229
Nuovi campi .....	230
Spostamento della logica .....	230
Ulteriori presentazioni .....	231
In questo capitolo .....	233

## Capitolo 24. Design by refactoring

Problema .....	235
Soluzione precedente .....	236
Il pattern Visitor .....	237
Esempio .....	237
Problema .....	240
Aggiunta di Visitor .....	240
Doppio dispatching con Visitor .....	245
Spostamento dei metodi .....	245
In questo capitolo .....	247

## Appendici

### A. Codice degli esempi

Applicare i pattern .....	251
Template e Factory Method .....	253
Decorator .....	255
Observer .....	257
Strategy .....	257
Pattern Half-Object .....	258
Lato server .....	258
Lato Client .....	261
Gestione di valute .....	265
Separate Domain from Presentation .....	268
Programma principale .....	268
Modello .....	270
View .....	271

### B. Riferimenti bibliografici .....

Indice analitico .....	283
------------------------	-----



# 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 [...].”

RICHARD P. GABRIEL

*Elementi 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 agli 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

manca 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 elementi 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 condividere 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.

