Prefazione

Una delle caramelle che preferiamo, qui in Danimarca, si chiama Ga-Jol, i cui intensi effluvi di liquirizia sono un toccasana per il nostro clima, così freddo e umido. Parte del nostro interesse per le caramelle Ga-Jol deriva dalla frase, fra il saggio e il faceto, stampata sulla linguetta di ogni scatolina. Questa mattina ho acquistato una confezione da due pacchetti di queste caramelle e una recava questo vecchio detto danese:

Ærlighed i små ting er ikke nogen lille ting.

"L'onestà nelle cose da poco, non è una cosa da poco." È un'ottima introduzione a quello che già intendevo dire in queste righe. Le piccole cose contano. Questo è un libro sui piccoli problemi il cui peso è tutt'altro che piccolo.

"Dio si ritrova nei dettagli", disse l'architetto Ludwig Mies van der Rohe. Questa citazione richiama le discussioni contemporanee sul ruolo dell'architettura nello sviluppo di software, e in particolare nel mondo Agile. Bob e io talvolta ci siamo scontrati proprio su questo argomento. E sì, Mies van der Rohe era attento all'utilità e all'eternità delle forme degli edifici basati su una grande architettura. D'altra parte, egli selezionò personalmente ogni singola maniglia di ogni singolo edificio che aveva costruito. Perché? Perché le piccole cose contano.

Nel "dibattito" in corso sullo sviluppo TDD (*Test-Driven Development*), Bob e io abbiamo scoperto di concordare sul fatto che l'architettura del software ha una collocazione importante nello sviluppo, sebbene abbiamo, probabilmente, idee differenti sull'esatto significato di questa affermazione. Tali cavilli non sono però particolarmente importanti, perché possiamo dare per scontato che i veri professionisti dedichino del tempo a riflettere e a pianificare il progetto fin dall'inizio. I concetti di fine anni Novanta sulla progettazione guidata unicamente dai test e dal codice sono ormai tramontati. Purtuttavia, l'attenzione al dettaglio è un elemento ancora più caratteristico della professionalità di qualsiasi grande visione. Innanzitutto, è proprio dalla cura delle piccole cose che i professionisti traggono competenza e fiducia per operare su quelle grandi. In secondo luogo, il piccolo difetto costruttivo, la portiera che non si chiude perfettamente o la mattonella leggermente disassata nel pavimento, o anche la scrivania in disordine, dissipano in un lampo l'immagine dell'intero insieme. Per questo parliamo di *codice pulito*.

In ogni caso, l'architettura è solo una delle metafore dello sviluppo di software e, in particolare, di quella parte del software che deve occuparsi di fornire il prodotto iniziale, nello

stesso modo in cui un architetto fornisce un edificio nuovo di zecca. In questi tempi di Scrum e Agile, il focus è sulla rapidità dell'uscita sul mercato. Vogliamo che la fabbrica del software possa operare a pieno regime. Queste sono le fabbriche umane: riflettere, considerare il fatto che essere programmatori significa operare su un prodotto preesistente o sulle richieste dell'utente, per creare un nuovo prodotto. La metafora della produzione incombe sempre più fortemente in questo modo di pensare. Gli aspetti produttivi dei costruttori d'auto giapponesi, della catena di montaggio, hanno ispirato molto Scrum.
Tuttavia, anche nel settore automobilistico, la parte principale del lavoro non sta tanto nella produzione, quanto nella manutenzione (o nell'evitarla il più possibile). Nel software, l'80 percento (se non di più) di quello che facciamo rientra nella categoria "manutenzione": attività di riparazione. Invece di adottare il tipico atteggiamento occidentale di concentrarci sulla realizzazione di buon software, dovremmo cominciare a ragionare più nei panni del "manutentore" di un'abitazione o del meccanico di auto. Che cosa dice il management giapponese a questo proposito?
All'incirca nel 1951, sulla scena giapponese nacque un approccio chiamato TPM, *Total Productive Maintenance*. Tale approccio si concentrava più sulla manutenzione che sulla produzione. Uno dei grandi pilastri dell'approccio TPM erano i cosiddetti Principi delle 5 "S". Si tratta di un insieme di *discipline*, un termine, quest'ultimo, scelto non a caso. Questi Principi delle 5 "S" sono alla base della progettazione "snella" (*lean*), un altro termine molto in voga in Occidente, e sempre più diffuso anche negli ambienti di sviluppo software. Questi principi sono imprescindibili. La buona pratica dello sviluppo software richiede proprio tale disciplina: attenzione, concentrazione e riflessione. Non si tratta solo del "fare", *solo del fare in modo* che la macchina produttiva possa operare a pieno regime. La filosofia delle 5 "S" riguarda i seguenti concetti.

- *Seiri*, organizzazione. È fondamentale sapere dove si trovano le cose (usando approcci come una corretta denominazione). Pensate che la denominazione degli identificatori non sia poi così importante? Leggete i prossimi capitoli.
- *Seiton*, ordine. Un vecchio proverbio recita: "Un posto per ogni cosa e ogni cosa al suo posto". Un frammento di codice deve trovarsi esattamente dove è previsto che debba trovarsi; se non si trova lì, fate in modo che lo sia.
- *Seiso*, pulizia. Tenete l'ambiente di lavoro libero da fili, olio, oggetti e materiali di scarto. Che cos'hanno da dire gli autori del libro sul codice commentato che ha il solo scopo di ricordare passaggi precedenti o desideri per il futuro? Che dovete sbarazzarvene.
- *Seiketsu*, standardizzazione. Tutti devono concordare sul modo in cui mantenere pulito l'ambiente di lavoro. Pensate che questo libro abbia qualcosa da dire sulla coerenza dello stile di programmazione e sulla scelta di pratiche comuni nel gruppo? Da dove vengono tali standard? Continuate a leggere.
- *Shutsuke*, disciplina. Significa adottare una precisa auto-disciplina, riflettere frequentemente sul proprio lavoro ed essere pronti a cambiare.

Se accettate la sfida (sì, perché di una vera sfida si tratta) di leggere e poi mettere in pratica questo libro, finirete sicuramente per comprendere e apprezzare quest'ultimo punto. Qui giungiamo, infine, alle radici della professionalità, in un'attività che dovrebbe considerare l'intero ciclo di vita di un prodotto. Se gestiamo le automobili (e i meccanismi in genere) con un approccio TPM, la manutenzione per guasti (ovvero dopo che i bug sono ormai emersi) diventa l'eccezione. Saliamo di un livello: ispezioniamo le

macchine quotidianamente e interveniamo sulle parti usurate prima che si rompano, o eseguiamo l'equivalente del classico "cambio dell'olio" ogni 10.000 km per prevenire le usure. Nel codice, dedichiamoci senza pietà al refactoring. Si può salire di un ulteriore livello, in quanto l'approccio TPM si è rinnovato oltre 50 anni fa: costruendo macchine che siano, già in partenza, di facile manutenzione. Far sì che il codice risulti leggibile è un passo importante per renderlo anche eseguibile. Le nuove tecniche, introdotte in ambiente TPM negli anni Sessanta, prevedono l'introduzione di una macchina interamente nuova o la sostituzione di quelle vecchie. Come ci esorta Fred Brooks, probabilmente dovremmo riscrivere da zero parti importanti del software all'incirca ogni sette anni, per eliminare la "sporcizia". Forse addirittura dovremmo ridurre questa costante temporale di Brooks, contandola invece in settimane, giorni o ore invece che in anni. È lì che si trovano i dettagli.

I dettagli recano dentro di loro una grande potenza, e vi è qualcosa di umile e profondo in questo approccio alla vita, come possiamo, come da stereotipo, attenderci da ogni approccio di origine orientale. Ma questo non è un atteggiamento solo orientale; anche i proverbi occidentali sono ricchi di ammonimenti in questo senso. La citazione che abbiamo appena scritto relativa al *Seiton* è stata usata da un pastore dell'Ohio, che considerava letteralmente l'ordine "come un rimedio per ogni tipo di malvagità". È che dire del *Seiso*? La pulizia è l'anticamera della divinità. Per quanto possa essere bella una casa, una tavola in disordine le toglie ogni splendore. E del *Shutsuke* a proposito delle piccole cose? Chi è onesto nelle piccole cose, lo è anche in quelle grandi. Questo significa anticipare: rifattorizzare oggi, con il giusto anticipo, prendendosi tempo per le successive "grandi" decisioni, invece di procrastinare. Un attimo di tempo ora ne fa risparmiare nove domani. Il mattino ha l'oro in bocca. Non rimandare a domani ciò che puoi fare oggi (quello era il senso originale della frase "con il giusto anticipo" nell'approccio "snello" prima che tutto cadesse sotto le sgrinfie dei consulenti software). E che dire del curare le cose piccole, l'impegno sul dettaglio in vista del risultato? Da piccoli semi nascono grandi alberi. E dell'integrare del lavoro preventivo nella routine quotidiana? Meglio prevenire che curare. Una mela al giorno toglie il medico di torno. L'approccio "clean code" rende onore alle radici più profonde della nostra cultura, o almeno alla nostra cultura così come era una volta, o come forse dovrebbe essere, e il tutto basato solo sull'attenzione al dettaglio.

Anche nella grande letteratura che parla di architettura troviamo righe a supporto di questi cosiddetti dettagli. Pensate alle maniglie di Mies van der Rohe. Questo è *Seiri*. Si tratta dell'attenzione al nome di ogni singola variable. Dovreste denominare ogni variabile con la stessa cura che dedichereste a un figlio.

Come sa chiunque abbia una casa, tali cure e tali raffinamenti non finiscono mai. L'architetto Christopher Alexander, padre del concetto di *pattern* e dei linguaggi dei *pattern*) considera ogni atto di progettazione come un piccolo atto di riparazione locale. E considera la realizzazione dei dettagli più fini della struttura l'unica competenza dell'architetto; le forme più grandi possono essere demandate ai pattern e la loro applicazione agli abitanti. La progettazione è una continua evoluzione, non solo quando aggiungiamo una nuova stanza alla casa, ma quando curiamo la tinteggiatura, quando sostituiamo i tappeti consumati o quando cambiamo il lavandino della cucina. La maggior parte delle attività reca in sé sentimenti analoghi. Nella nostra ricerca di qualcuno cui ascrivere l'idea che Dio si trovi in dettagli, ci troviamo in buona compagnia, con lo scrittore francese dell'Ottocento Gustav Flaubert. Il poeta francese Paul Valery ci confessa che una poesia

non è mai terminata e richiede un lavoro continuo; smettere di lavorarvi sarebbe una resa. Tale cura per i dettagli è comune in tutte le attività che puntano all'eccellenza. Pertanto, forse qui non diciamo nulla di nuovo, ma nella lettura di questo libro verrete esortati ad adottare alcune buone pratiche alle quali potete aver rinunciato per stanchezza o per privilegiare la spontaneità, e quindi a "rispondere ai cambiamenti".

Sfortunatamente, in genere non consideriamo tali riflessioni come elementi fondanti dell'arte della programmazione. Abbandoniamo il codice a se stesso troppo presto, non perché sia terminato, ma perché le nostre priorità riguardano più l'aspetto esteriore che la sostanza del prodotto che forniamo.

Questa superficialità ha un costo finale: alla fine il problema salta sempre all'occhio. La ricerca, in ambito professionale e accademico, si umilia limitandosi a esortare alla pulizia del codice. Ai tempi in cui lavoravo presso i Bell Labs Software Production Research (dunque in *produzione*) avevamo alcune norme condivise, una delle quali suggeriva che uno stile di indentazione coerente era uno degli indicatori statisticamente più significativi della ridotta densità di bug. Si supponeva che l'architettura o il linguaggio di programmazione o qualche altro concetto di alto livello fosse da solo responsabile della qualità. In quanto persone che dovevano la loro professionalità alla capacità di impiegare gli strumenti disponibili e a ottimi metodi di progettazione, ci sentivamo insultati dal valore che queste macchine da produzione, i codificatori, aggiungevano tramite la mera applicazione di uno stile di indentazione coerente. Per citare il mio blocco di appunti di 17 anni fa, tale stile distingue l'eccellenza dalla mera competenza. La visione giapponese comprende il valore fondamentale del lavoratore e, ancora di più, dei sistemi di sviluppo che si fondano sulle semplici azioni quotidiane di tali lavoratori. La qualità è il risultato di milioni di singoli e disinteressati atti di cura, non solo di un qualche metodo di sviluppo piovuto dal cielo. Il fatto che questi atti siano semplici non significa che siano banali, e poi resta da dimostrare che siano davvero tanto semplici. Sono piccoli, ma ciononostante sono responsabili della grandezza, ma di più, della bellezza, di ogni attività umana. Ignorarli significa non essere pienamente umani.

Naturalmente, sono sempre convinto che si debba pensare su una scala più ampia, e in particolare sono convinto del valore degli approcci all'architettura basati su una profonda conoscenza del dominio e sull'usabilità del software. Il libro non tratta questo argomento o, almeno, non lo tratta direttamente. Questo libro ha in sé un messaggio più sottile, la cui profondità non dovrebbe essere sottovalutata. Rientra nel filone di alcuni personaggi chiave della progettazione del software, come Peter Sommerlad, Kevlin Henney e Giovanni Asproni. I loro mantra sono "The code is the design" e "Simple code". Anche se dobbiamo sempre ricordare che l'interfaccia è il programma, e che le sue strutture dicono molto della struttura del nostro programma, è fondamentale adottare continuamente la semplice affermazione che la struttura vive dentro il codice. E mentre la rielaborazione, secondo la metafora della produzione, introduce nuovi costi, la rielaborazione della struttura aumenta il valore. Dovremmo considerare il nostro codice come una bellissima espressione articolata di un nobile impegno di progettazione: progettazione intesa come un processo, non come un punto terminale statico. È nel codice che emergono le metriche d'architettura dell'accoppiamento e della coesione. Se sentite Larry Constantine parlarvi di accoppiamento e coesione, lo sentirete parlare di codice, non di concetti astratti in stile UML. Richard Gabriel ci dice nel suo saggio *Abstraction Descant* che l'astrazione è il male. Il codice è contro il male e il *codice pulito* è semidivino.

Tornando alla mia scatolina di Ga-Jol, penso che sia importante notare che il piccolo proverbio danese ci esorta non solo a fare attenzione alle piccole cose, ma anche a essere onesti nelle cose di poco valore. Significa essere onesti con il codice, onesti coi colleghi sullo stato del nostro codice e, soprattutto, essere onesti con noi stessi a proposito del nostro codice. Abbiamo fatto del nostro meglio per "lasciare il luogo più pulito di come l'abbiamo trovato"? Abbiamo eseguito il refactoring del nostro codice prima di fornirlo? Questi non sono dettagli ininfluenti, ma concetti che si situano proprio al centro dei valori Agile. Si tratta di una pratica consigliata in Scrum che il refactoring entri nel concetto stesso di "Finito!". Né l'architettura né il codice pulito si basano sulla perfezione, ma solo sull'onestà e sul fatto di dare il proprio meglio. Errare è umano; perdonare è divino. In Scrum, rendiamo tutto visibile. Siamo onesti sullo stato del nostro codice perché il codice non è mai perfetto. Diventiamo così più pienamente esseri umani, più meritevoli del divino e più vicini alla grandezza dei dettagli.

Nella nostra professione, abbiamo disperatamente bisogno di tutto l'aiuto possibile. Se un pavimento pulito riduce gli incidenti e una buona organizzazione degli attrezzi incrementa la produttività, sono assolutamente favorevole. Questo libro è la migliore espressione pragmatica dei principi "snelli" allo sviluppo di software che io abbia mai visto. Non mi aspettavo niente di meno da questo piccolo gruppo di personaggi, che per anni si sono sforzati, insieme, non solo a essere migliori, ma anche di divulgare le proprie competenze, nel loro ambito, tramite opere come quella che tenete in mano. Ora che l'editore mi ha inviato questo manoscritto sento che il mondo è un posto un po' migliore.

Ora che ho terminato questo esercizio, scritto con i miei migliori auspici, è tempo di mettere in ordine la mia scrivania.

James O. Coplien
Mørdrup, Danimarca