Capitolo 1

La pozza di bitume

Een schip op het strand is een baken in zee.
[Una nave sulla spiaggia è un faro per il mare.]
– Proverbio olandese

Non c'è scena della preistoria tanto vivida come quella di grandi animali che si dibattono per salvarsi la vita in pozze di bitume. Nella nostra fantasia vediamo dinosauri, mammuth e tigri dai denti a sciabola che lottano contro la presa del bitume. Quanto più si dibattono, tanto più vengono intrappolati e non esiste animale tanto forte o tanto abile da non finire per affondare.

La programmazione di grandi sistemi nell'ultimo decennio è stata una pozza di bitume di questo genere e molti animali grandi e potenti vi sono rimasti intrappolati. Quasi tutti ne sono emersi con sistemi funzionanti, ma pochi rispettando obiettivi, calendari e budget. Grandi e piccoli, massivi o ridotti all'osso, team su team sono rimasti impegolati in questa pozza. Non sembra che la difficoltà sia generata da qualcosa in particolare, ma l'accumularsi di fattori simultanei e interagenti rende il movimento sempre più lento. Tutti sembra siano rimasti sorpresi dall'apparente vischiosità del problema ed è difficile discernerne la natura. Per risolvere il problema, però, dobbiamo cercare di capirlo.

Cominciamo quindi identificando l'arte della programmazione di sistema, le gioie e i dolori che l'accompagnano.

Il prodotto di sistemi di programmazione

Qualche volta sui giornali capita di leggere come due programmatori in un garage ristrutturato abbiano costruito un programma importante

che supera quanto di meglio abbiano prodotto grandi team. Ogni programmatore, per parte sua, è pronto a credere a queste storie, perché sa che potrebbe costruire *qualsiasi* programma molto più rapidamente rispetto a quei 1000 enunciati/anno dichiarati per i team industriali.

Perché allora i team industriali di programmazione non sono stati sostituiti da appassionate coppie in qualche garage? Bisogna tenere conto di *che cosa* viene prodotto.

Nel quadrante superiore sinistro della Figura 1.1 c'è un *programma*. È completo in sé, pronto per essere eseguito dall'autore sul sistema su cui lo ha sviluppato. *Quella* è la cosa che viene normalmente prodotta nei garage, ed è l'oggetto che il singolo programmatore utilizza per stimare la produttività.

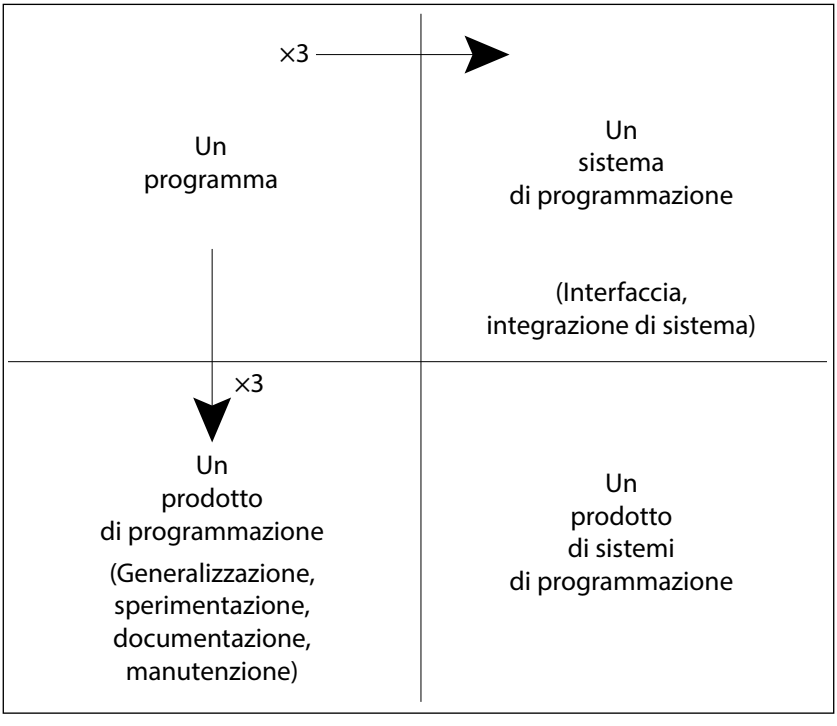

Figura 1.1 *Evoluzione del prodotto di sistemi di programmazione.*

Esistono due modi per convertire un programma in un oggetto più utile, ma più costoso, e sono rappresentati dalle due linee che dividono il diagramma in quadranti.

Se ci si sposta verso il basso e si supera il confine orizzontale, un programma diventa un *prodotto di programmazione*. Questo è un

programma che può essere eseguito, testato, riparato ed esteso da chiunque. È utilizzabile in molti ambienti operativi, per molti insiemi di dati. Per diventare un prodotto di programmazione usabile in generale, un programma deve essere scritto in una forma generalizzata. In particolare, il dominio e la forma degli input devono essere generalizzati per quanto l'algoritmo di base ragionevolmente consenta. Poi il programma deve essere sottoposto ampiamente a test, in modo che ci si possa fidare del suo funzionamento. Questo significa che bisogna preparare, eseguire e registrare un banco corposo di casi di test, che esplorino il dominio di input e ne sondino i confini. Infine, perché un programma venga promosso al rango di prodotto di programmazione, deve avere una documentazione completa, in modo che chiunque possa usarlo, ripararlo ed estenderlo. Come regola empirica, stimo che un prodotto di programmazione costi almeno tre volte quanto un programma, sottoposto a debug, che svolga la stessa funzione.

Se ci si sposta a destra e si supera il confine verticale, un programma diventa un componente di un *sistema di programmazione*: questo è un insieme di programmi che interagiscono, coordinati nelle loro funzioni e ben disciplinati nel loro formato, in modo che il loro assemblaggio costituisca un apparato unico per svolgere compiti di grandi dimensioni. Per diventare un componente di un sistema di programmazione, un programma deve essere scritto in modo che ogni input e ogni output si conformino, per sintassi e semantica, a interfacce definite con precisione. Il programma inoltre deve essere progettato in modo da usare solo un bacino prestabilito di risorse: spazio di memoria, dispositivi di input e output, tempo macchina. Infine, il programma deve essere sottoposto a test con altri componenti del sistema, in tutte le combinazioni previste. Questi test devono essere molto ampi, perché il numero dei casi cresce in modo combinatorio. È un procedimento che richiede molto tempo, perché possono emergere errori infidi dalle interazioni inattese di componenti già sottoposti singolarmente a debug. Un componente di un sistema di programmazione costa almeno il triplo di un programma autonomo che svolga la stessa funzione. Il costo può essere anche maggiore, se il sistema ha molti componenti.

Nel quadrante inferiore destro della Figura 1.1 si trova il *prodotto di sistemi di programmazione*, che si distingue dal programma semplice in tutti i modi precedenti. Costerà nove volte tanto, ma è l'oggetto veramente utile, il prodotto a cui mira la maggior parte delle iniziative di programmazione di sistema.

Le gioie del mestiere

Perché programmare è divertente? Quali delizie può attendersi come ricompensa chi pratica questa attività?

In primo luogo, la pura gioia di creare qualcosa. Come un bambino è deliziato dalla sua torta di fango, l'adulto prova piacere nel costruire cose, in particolare se si tratta di cose di sua progettazione. Penso che questa soddisfazione debba essere un'immagine della soddisfazione di Dio nella creazione, un piacere che si mostra nella differenza e nella novità di ogni foglia e di ogni fiocco di neve.

In secondo luogo, il piacere di creare cose che sono utili per altri. Nel profondo, vogliamo che gli altri usino il nostro lavoro e lo trovino utile. Da questo punto di vista il sistema di programmazione non è sostanzialmente diverso dal primo portapenne di argilla del bambino "per l'ufficio di papà".

In terzo luogo c'è il fascino di foggiare oggetti complessi, simili a rompicapo, fatti di parti mobili fra loro intrecciate, e vederli funzionare in cicli delicati, mostrando le conseguenze di principi che vi sono stati incorporati sin dall'inizio. Il computer programmato ha tutto il fascino del meccanismo del flipper o del jukebox, portato all'estremo.

In quarto luogo, la gioia di apprendere continuamente, che nasce dalla natura non ripetitiva delle attività che si svolgono. In un modo o nell'altro, il problema è sempre nuovo e chi lo risolve impara qualcosa: a volte qualcosa di pratico, a volte di teorico, a volte di entrambe le nature.

Infine, c'è il piacere di lavorare con un mezzo tanto plasmabile. Il programmatore, come il poeta, lavora con qualcosa che si allontana solo di poco dal puro pensiero. Costruisce i suoi castelli in aria, dall'aria, creando con l'esercizio dell'immaginazione. Pochi mezzi di creazione sono altrettanto flessibili, tanto facili da rifinire e rielaborare, tanto facilmente in grado di dare realtà a grandiose strutture concettuali. (Come vedremo più avanti, questa plasmabilità ha anche i suoi problemi.)

Tuttavia il costrutto "programma", a differenza delle parole del poeta, è reale, nel senso che si muove e svolge un lavoro, produce risultati visibili distinti dal costrutto stesso. Stampa risultati, disegna figure, produce suoni, sposta bracci meccanici. La magia del mito e delle leggende è diventata realtà nel nostro tempo. Si scrive su una tastiera l'incantesimo giusto e uno schermo prende vita, mostrando cose che non ci sono mai state né mai avrebbero potuto esserci.

Programmare quindi è divertente perché gratifica i desideri creativi presenti in profondità dentro di noi e delizia sensibilità che abbiamo in comune con tutti gli esseri umani.

I dolori del mestiere

Non tutto è delizia, però, e conoscere quali siano i dolori intrinseci rende più facile tollerarli quando fanno la loro comparsa.

Come prima cosa, bisogna fare tutto alla perfezione. Il computer ricorda la magia delle leggende anche sotto questo aspetto. Se un carattere o una pausa dell'incantesimo non sono esattamente nella forma giusta, la magia non funziona. Gli esseri umani non sono abituati a essere perfetti, e pochi campi dell'attività umana richiedono la perfezione. Adeguarsi al requisito della perfezione penso sia la parte più difficile dell'imparare a programmare[1].

Poi, sono altre le persone che fissano gli obiettivi, mettono a disposizione le risorse e forniscono le informazioni. Raramente ciascuno controlla le circostanze del proprio lavoro, o anche il proprio obiettivo. In termini di management, l'autorità del singolo non è sufficiente per la sua responsabilità. Sembra che in tutti i campi, però, i lavori in cui si fanno cose non abbiano mai un'autorità formale commisurata alla responsabilità. Nella pratica, si acquisisce un'autorità reale (anziché formale) in base allo slancio stesso del risultato raggiunto.

La dipendenza dagli altri in un caso è particolarmente dolorosa per il programmatore di sistema: deve dipendere dai programmi di altre persone. Questi spesso sono mal progettati, implementati in modo inadeguato, presentati in forma incompleta (senza codice sorgente o casi di test) e documentati in modo insufficiente. Perciò deve dedicare ore a studiare e riparare cose che, in un mondo ideale, sarebbero complete, disponibili e usabili.

Il dolore successivo deriva dal fatto che progettare grandi concetti è divertente, ma individuare piccoli e noiosi bug è solo lavoro. Ogni attività creativa è accompagnata da terribili ore di lavoro noioso e scrupoloso, e la programmazione non fa eccezione.

Poi si scopre che il debug ha una convergenza lineare o, peggio, ci si aspetta una sorta di andamento quadratico quando ci si avvicina alla fine. Così i test si trascinano e l'individuazione degli ultimi bug difficili richiede più tempo di quella dei primi.

L'ultimo dolore, e a volte l'ultima goccia, è che il prodotto su cui ci si è affannati tanto a lungo al momento del completamento (o anche prima) appare obsoleto. Già colleghi e concorrenti stanno perseguendo idee nuove, idee migliori. Già la sostituzione del parto della nostra mente non solo è stata concepita, ma anche messa a calendario.

Questo sembra sempre peggio di quel che è realmente. Il prodotto nuovo e migliore in generale non è *disponibile* quando si completa il proprio; se ne parla soltanto. Anche quello richiederà mesi di sviluppo. La tigre reale non è mai una degna avversaria della tigre di carta,

a meno che non si voglia un uso effettivo; a quel punto le virtù della realtà hanno una soddisfazione tutta propria.

Ovviamente la base tecnologica su cui si costruisce è *sempre* in movimento. Non appena si "congela" un progetto, diventa obsoleto, per quanto riguarda i suoi concetti. L'implementazione di prodotti reali comporta messa in fase e quantizzazione. L'obsolescenza di un'implementazione va misurata rispetto ad altre implementazioni esistenti, non in base a concezioni non realizzate. La sfida e la missione sono trovare soluzioni reali a problemi reali, in base a calendari effettivi con le risorse disponibili.

Questa dunque è la programmazione, sia una pozza di bitume in cui sono affogate molte iniziative, sia un'attività creativa con gioie e dolori tutti propri. Per molti, le gioie compensano abbondantemente i dolori, e per loro il resto di questo libro cercherà di gettare qualche passerella al di sopra della pozza di bitume.