Capitolo 1

Vivere in Produzione

Hai lavorato sodo sul tuo progetto. Ti sembra che finalmente tutte le funzionalità siano al loro posto e la maggior parte di esse ha anche passato i test. Puoi tirare un sospiro di sollievo. Hai finito.

Ne sei proprio sicuro?

Quel "funzionalità al loro posto" significa "pronte per la Produzione"? Il sistema è davvero pronto per essere distribuito? Può funzionare in un ambiente di lavoro ed è pronto ad affrontare senza di te le orde di utenti veri del mondo reale? Stai già iniziando a provare quella sensazione di fallimento e "pregustando" le telefonate allarmate e i messaggi che arrivano a tarda sera? Sì: lo sviluppo è molto di più che aggiungere semplicemente a un programma tutte le funzionalità richieste.

La progettazione del software, così come viene insegnata oggi, è drammaticamente incompleta. Dice solo quello che i sistemi *dovrebbero* fare. E quindi non si occupa del contrario: di quello che i sistemi *non dovrebbero* fare. Non dovrebbero andare in crash, bloccarsi, perdere dati, violare la privacy, far perdere denaro al cliente, distruggere la vostra azienda e, possibilmente, neanche uccidere i vostri clienti.

Troppo spesso, i team di progettazione mirano a superare i test sottoposti loro dal dipartimento di controllo qualità (QA) invece di puntare alla vera vita del software, in Produzione. Quello che voglio dire è che probabilmente la maggior parte del vostro lavoro si concentra sul superamento dei test. Ma il testing – anche se agile, pragmatico e automatizzato – non è sufficiente a dimostrare che il software sia pronto per affrontare il mondo reale. Gli stress e le tensioni cui verrà esposto, con tutte le follie degli

In questo capitolo

utenti reali, un traffico di dati che può attraversare il mondo intero e con autori di virus che operano da paesi che magari non avete mai nemmeno sentito nominare possono andare ben oltre quello che potrete mai sperare di sottoporre a test.
Ma in primo luogo, dovete accettare il fatto che, nonostante i vostri migliori propositi, le "brutte cose" possono accadere. Naturalmente è sempre bene impedirle, quando possibile. Ma può essere addirittura fatale supporre di aver previsto ed eliminato tutte le potenziali eventualità negative. Al contrario, dovrete agire e prevenire quanto più potete, e fare in modo che il sistema, nel suo complesso, possa reggere a qualsiasi trauma imprevisto e grave che possa verificarsi.

Puntare al bersaglio giusto

La maggior parte del software viene progettata per il laboratorio di sviluppo o per i tester del reparto QA. Viene progettato e costruito per superare test come: "Il nome e il cognome del cliente sono necessari, mentre l'iniziale centrale è facoltativa". Ma in tal modo punta solo a sopravvivere al regno artificiale della QA, non al mondo reale della Produzione.
La progettazione del software oggi somiglia al design di automobili nei primi anni Novanta: è scollegato dal mondo reale. Le auto, progettate esclusivamente nel comfort del laboratorio, sembravano magnifiche nei modelli e nei sistemi CAD. Vetture dalle curve aerodinamiche brillano di fronte ai grandi ventilatori che inviano un perfetto flusso laminare. I progettisti che popolano questi spazi così ricchi di serenità producono progetti eleganti, sofisticati, intelligenti, ma fragili, insoddisfacenti e, in ultima analisi, di breve durata. La maggior parte dell'architettura e della progettazione del software vede la luce in ambienti altrettanto puliti e lontani dalla realtà.
Davvero la vorreste una macchina bellissima, ma che trascorre più tempo dal meccanico che sulla strada? Certo che no! Probabilmente vorrete possedere una macchina fatta per il mondo reale. Volete una macchina progettata da qualcuno che sa che i cambi d'olio vengono fatti *sempre* con 3.000 chilometri di ritardo, che le gomme devono funzionare altrettanto bene da nuove e all'ultimo millimetro di battistrada e che, certamente, a un certo punto debba sopportare una gran pedata sui freni mentre tenete un hamburger in una mano e il telefono nell'altra.
Quando il nostro sistema passa il vaglio della QA, possiamo davvero dire che sia pronto per la Produzione? Il semplice passaggio della QA ci dice poco circa l'idoneità del sistema nei prossimi tre-dieci anni di vita. Potrebbe rivelarsi la Toyota Camry del software e accumulare migliaia di ore di utilizzo continuo. Oppure potrebbe finire per essere la Chevy Vega (una vettura che faticò già nel corso dei test interni) o la Ford Pinto (una macchina incline a saltare in aria, in particolari condizioni). È impossibile che un paio di giorni o anche di settimane di test possa dire ciò che avverrà nei prossimi anni.
I progettisti che lavorano in Produzione hanno a lungo perseguito la "progettazione per la producibilità": l'approccio ingegneristico alla progettazione di prodotti che possano essere fabbricati a basso costo e con elevata qualità. Ma prima di oggi, i progettisti e i fabbricanti di prodotti vivevano in mondi diversi. I disegni appesi alla parete della Produzione comprendevano viti irraggiungibili, parti difficilmente riconoscibili e parti realizzate su ordinazione quando bastava impiegare componenti da scaffale. Inevitabilmente, ciò ha portato a una bassa qualità e a un elevato costo di produzione.

Oggi siamo in una situazione simile. Finiamo per non riuscire a terminare il nuovo sistema, perché stiamo costantemente rispondendo alle chiamate di supporto per l'ultimo progetto che, pronto solo a metà, abbiamo immesso a forza sul mercato. Per noi l'analogo della "progettazione per la producibilità" è la "progettazione per la Produzione". Noi non mandiamo progetti ai produttori, ma inviamo il software finito ai clienti. Abbiamo bisogno di progettare i singoli sistemi software, ma anche l'intero ecosistema di sistemi interdipendenti, in modo che operino a basso costo e a elevata affidabilità.

La portata della sfida

Ai tempi, facili e tranquilli, dei sistemi client/server, la base di utenti di un sistema si sarebbe misurata in decine o centinaia di unità con, al massimo, una dozzina di utenti concorrenti. Oggi vediamo regolarmente conteggi di utenti attivi ben superiori e con una popolazione che si estende su interi continenti. E non intendo l'Antartide o l'Australia! Il primo social network, oggi, conta miliardi di utenti e non sarà l'ultimo.
Sono aumentate anche le richieste in termini di uptime. Mentre il famoso tempo di uptime a "cinque nove" (99,999 percento) era un tempo tipico per i sistemi a mainframe e per i suoi operatori, oggi anche i siti di commercio di articoli da giardino richiedono una disponibilità 24/7/365 (questa sigla mi ha sempre infastidito: da ingegnere, mi aspetterei che sia "24/365" oppure "24/7/52"). Chiaramente, abbiamo compiuto enormi passi avanti anche nel considerare la scalabilità del software costruito oggi; ma aumentando la portata e la scala dei nostri sistemi, scopriamo sempre nuove occasioni di guasto, ambienti sempre più ostili e una tolleranza sempre minore ai difetti.
La crescente portata di questa sfida (costruire software veloce che sia economico da realizzare, efficace per gli utenti ed economico anche nell'utilizzo) richiede continui raffinamenti dell'architettura e delle tecniche di progettazione. Una progettazione appropriata per i piccoli siti web di Wordpress fallisce clamorosamente se applicata a sistemi distribuiti a larga scala, transazionali. E vedremo proprio alcuni di questi clamorosi insuccessi.

Milione di dollari più, milione di dollari meno…

La posta in gioco è alta: il successo del vostro progetto, le quotazioni delle azioni o la ripartizione degli utili, la sopravvivenza della vostra azienda e il vostro stesso lavoro. I sistemi costruiti per la QA spesso richiedono continue spese, sotto forma di costi operativi, tempi di inattività e manutenzione del software, tanto da non consentire loro di raggiungere, mai, la redditività, per non parlare di un saldo netto positivo per l'azienda (raggiunto solo dopo che i profitti generati dal sistema rimborsano i costi sostenuti per la sua realizzazione). Questi sistemi mostrano una bassa disponibilità, con perdite dirette in profitti perduti e perdite indirette in termini di danni al marchio.
Durante la frenetica corsa dello sviluppo di un progetto, è fin troppo facile prendere decisioni che ottimizzano i costi di sviluppo a scapito dei costi operativi. Questo ha senso solo nel contesto di un team che miri a un budget fisso e a rispettare la data di consegna. Ma nel contesto dell'organizzazione che acquista quel software, si tratta di un cattivo investimento. I sistemi trascorrono la maggior parte della loro vita in operatività e non nella fase di sviluppo (almeno quelli che non vengono annullati o scartati). Evitare una

volta un determinato costo di sviluppo, per poi incorrere in un costo operativo ricorrente non ha senso. In realtà, la decisione opposta ha molto più senso dal punto di vista finanziario. Immaginate che il vostro sistema richieda cinque minuti di downtime a ogni release. Ci si aspetta che il sistema abbia un orizzonte di vita di cinque anni, con release mensili (la maggior parte delle aziende vorrebbe far uscire più release all'anno, ma non voglio esagerare). Potete calcolare il costo previsto dei tempi di inattività, scontato per il valore temporale del denaro. Probabilmente la cifra sarà dell'ordine di un milione di dollari (300 minuti di downtime a un costo molto modesto di 3.000 dollari al minuto). Supponiamo ora che possiate investire 50.000 dollari per creare una sequenza di compilazione e un processo di distribuzione che eviti di incorrere in tempi di inattività durante le release. Come minimo, scongiurereste una perdita da un milione di dollari. È molto probabile che questo vi permetterà anche di aumentare la frequenza di deployment e di catturare una maggiore quota di mercato. Ma limitiamoci ai guadagni diretti, per ora. Alla maggior parte dei direttori finanziari non dispiacerebbe affatto autorizzare una spesa che renda il 2.000 percento in termini di ritorno degli investimenti!
Le decisioni relative alla progettazione e all'architettura sono anche di natura finanziaria. Queste scelte devono essere compiute con un occhio ai costi di implementazione e l'altro ai costi a valle. La fusione dei punti di vista tecnico e finanziario è uno dei più importanti temi ricorrenti di questo libro.

Usare la forza

Le decisioni iniziali sono quelle che hanno il maggiore impatto sulla forma finale del sistema, perché possono essere le più difficili da invertire. Queste decisioni iniziali sui confini definiti nel sistema e sulla sua scomposizione in sottosistemi finiscono per cristallizzarsi nella struttura del team, nell'allocazione dei finanziamenti, nella struttura della gestione del programma e perfino in termini di allocazione dei tempi. Le assegnazioni ai team costituiscono la prima bozza dell'architettura. Ironia della sorte, queste decisioni così preliminari sono anche le meno informate. All'inizio il team per lo più ignora quale sarà la struttura finale del software e tuttavia è qui che devono essere prese alcune delle decisioni più irrevocabili.
Mi dichiaro, qui e ora, fautore dello sviluppo agile. L'enfasi sulla consegna anticipata e sui miglioramenti incrementali fa sì che il software entri rapidamente in Produzione. Dal momento che la Produzione è l'unico luogo in cui si può imparare come il software reagirà agli stimoli del mondo reale, sostengo un approccio che consenta di iniziare il processo di apprendimento il più presto possibile. Anche su progetti agili, è meglio che le decisioni vengano prese con lungimiranza. Sembra proprio che il progettista debba "usare la forza" per prevedere il futuro, al fine di selezionare la struttura più solida. Poiché alternative differenti spesso hanno costi simili di implementazione, ma costi radicalmente differenti nel ciclo di vita, è importante considerare gli effetti di ogni decisione in termini di disponibilità, capacità e flessibilità. Vi mostrerò gli effetti che si producono a valle per decine di alternative progettuali, con esempi concreti di approcci positivi e nocivi. Questi esempi sono tratti da sistemi reali, sui quali ho lavorato io stesso. E la maggior parte di essi mi è costata parecchie ore di sonno.

Architettura pragmatica

Sono due le attività, divergenti, che rientrano nel termine *architettura*. Un tipo di architettura punta verso i livelli più elevati di astrazione, che risultino maggiormente portatili su più piattaforme e meno legati ai dettagli dell'hardware, delle reti, degli elettroni e dei fotoni. La forma estrema di questo approccio si traduce in una "torre d'avorio": una camera asettica "alla Kubrick" abitata da guru completamente avulsi dal mondo e piena di grafici e frecce, tracciati su ogni parete. I decreti emessi dalla torre d'avorio discendono sui programmatori al lavoro. "Che il middleware sia JBoss, ora e per sempre!", "Che le interfacce utente siano costruite con Angular 1.0!", "Tutto ciò che è stato, che è e che sarà vive in Oracle!", "Non commettere atti impuri con Ruby!". Se vi è mai capitato di digrignare i denti, mentre programmavate qualcosa secondo gli "standard aziendali" imposti mentre sarebbe dieci volte più facile farlo con qualche altra tecnologia, allora sapete già che cosa significa essere vittime di un architetto da torre d'avorio. Vi garantisco che un architetto che non si preoccupi di ascoltare i programmatori del team non si preoccupa neanche di ascoltare gli utenti. Il risultato lo conosciamo: gli utenti sono addirittura felici quando il sistema si blocca, perché almeno possono smettere di usarlo per un po'.

Al contrario, un'altra razza di architetto non solo sta fianco a fianco con i programmatori, ma è uno di loro. Questo tipo di architetto non esita a sacrificare un po' un'astrazione o a disfarsene, se scopre che non è adatta. Questo architetto, pragmatico, è più probabile che ami discutere di questioni come l'utilizzo della memoria, i requisiti in termini di CPU, le esigenze di larghezza di banda e i vantaggi e svantaggi dell'hyperthreading e del CPU binding.

L'architetto da torre d'avorio si crogiola in una visione mistica dello stato finale della sua creazione, che vanta una perfezione cristallina, laddove l'architetto pragmatico pensa costantemente in termini di dinamica del cambiamento. "Come possiamo eseguire un deployment senza riavviare tutto quanto?", "Quali metriche abbiamo bisogno di raccogliere e come le analizziamo?", "Quale parte del sistema ha più bisogno di essere migliorata?". Quando l'architetto da torre d'avorio ha terminato, il sistema non ammette più alcun miglioramento; ogni parte sarà perfettamente adatta al suo ruolo. Al contrario, la creazione di un architetto pragmatico prevede che ogni componente sia sufficientemente "buono" per i fattori di stress attuali; e l'architetto sa quali componenti dovranno essere rielaborati, a seconda di come i fattori di stress cambieranno nel corso del tempo. Se siete già architetti pragmatici, allora qui troverete capitoli e capitoli pieni di potenti munizioni per voi. Se siete architetti da torre d'avorio – e non avete ancora smesso di leggere – allora questo libro potrebbe invogliarvi a scendere di alcuni livelli di astrazione, per tornare in contatto con quella intersezione che è vitale per il software, l'hardware e gli utenti: la sua vita in Produzione. Voi, i vostri utenti e la vostra azienda saranno molto più felici quando finalmente arriverà il momento del rilascio!

Conclusioni

Il software esprime tutto il suo valore solo in Produzione. Il progetto di sviluppo, il testing, le fasi di integrazione e pianificazione… tutto ciò che viene prima della Produzione è

solo un preludio. Questo libro si occupa della vita in Produzione, dalla release iniziale alla crescita continua e all'evoluzione del sistema. La prima parte di questo libro si occupa della stabilità. Per dare un'idea del tipo di problemi legati al fatto che dobbiamo evitare che un software vada in crash, iniziamo occupandoci di un bug che ha tenuto a terra un'intera compagnia aerea.