Introduzione

Costruire dal niente

"Mi piace perché è come costruire dal niente": questa frase l'ho sentita pronunciare a un bambino delle scuole elementari mentre giocava con Scratch, la piattaforma creata dal MIT per insegnare i concetti della programmazione come se fossero un piccolo puzzle da risolvere che poi prende vita, si anima. L'idea di animare qualcosa dal niente però non è nuova. Tra il 1908 e la metà degli anni Novanta l'editore del "Corriere della Sera" ha pubblicato un inserto periodico per bambini molto diffuso in tutta la penisola [Cor12]. Il "Corriere dei Piccoli" nasceva come settimanale a fumetti e raccontava le storie di personaggi di ogni genere alle prese con avventure scientifiche, drammi di attualità politica, imprese storiche e gesta eroiche.

Negli anni alcuni protagonisti di queste storie sono entrati a far parte della cultura popolare italiana, diventando noti al pubblico di ogni età: una permeabilità generazionale che solo le grandi penne si guadagnano. Sergio Tofano con *Il Signor Bonaventura*, per esempio, ma anche Bruno Angoletta con *Marmittone* o Italo Calvino con *Marcovaldo*. All'inizio degli anni Trenta, a firma di Giovanni Manca, arrivano sulle colonne del settimanale le storie di un vecchio professore, occhialuto e dal naso rosso: è il professor *Pier Cloruro de' Lambicchi*. Un nevrotico, impacciato, sfortunato ma geniale inventore, celebrità per aver inventato l'*arcivernice*. Questa invenzione straordinaria dava proprio la possibilità di costruire dal niente: trasferire i pensieri che frullavano nella testa dell'artista fino alla realtà in maniera quasi impeccabile. Si trattava di una speciale tintura in grado di far prendere vita ai quadri e ai personaggi che vi comparivano. Il meccanismo era questo: il professore pensava a un personaggio e lo disegnava sulla tela, poi con una passata di arcivernice (stesa come fosse una tempera) il fortunato prendeva vita. Bastava abbozzare un disegno per vederlo trasformato nell'eroe di turno, che si animava per un'ora. Il professore era un tipo un po' pasticcione e le conseguenze per il mondo reale erano sì buone, ma più spesso disastrose.

NOTA

La metafora dell'*arcivernice* l'ho sentita per la prima volta dal professor Angelo Raffaele Meo, pioniere dell'informatica in Italia [Meo13]; così presentava il software in una delle sue conferenze alla fine degli anni Duemila.

L'ultimo numero del "Corrierino" fu stampato quando la testata aveva quasi novant'anni di vita, nell'agosto 1995. La seconda metà di quel decennio fu un momento di accelerazione per l'espansione di Internet, con l'arrivo delle prime connessioni analogiche a 56K nelle case.
Appartengo alla prima generazione che ha imparato a leggere su uno schermo, anche se ancora non quello di uno smartphone: ho avuto a disposizione un computer fisso. Era uno strumento davvero curioso e anche io subivo il fascino di un'*arcivernice*: i linguaggi di programmazione (il C in particolare, su un Intel i386DX, con ancora il tasto *turbo*). Ero attratto dalla possibilità di poter digitare qualche riga di testo e poi con un tasto vederle animate, proprio come quel bambino con Scratch. Anche se il software non ha lo stesso carattere forte e indipendente dei personaggi del *Lambicchi*, i computer hanno la stessa possibilità di dar vita a sistemi che attraverso l'hardware hanno un effetto nel mondo reale. Il codice sorgente è un disegno che contiene le regole, che descrive come funziona e decide un sistema; il compilatore è l'ingrediente che con una passata trasforma il tutto in un oggetto confezionato e animabile. Più il quesito da risolvere è consistente, più le regole che servono a risolverlo sono numerose e devono assumere dettagli, ramificazioni, entrare in ambiti di studio diversi.
Sono *sistemi*: insiemi animati di parti, montati e collegati tra loro. Pensate ai mattoncini di plastica Lego e all'infinita varietà di tipi, modelli e colori diversi. Piattaforme, ingranaggi, ruote, eliche, omini, porte, finestre! Ognuna è una parte con finalità e regole di interazione differenti; magari alcune sono in comune, ma si collegano con incastri diversi. Montate assieme possono diventare un treno, un'astronave, una gru, una casa. Basta un'idea per partire e costruire qualcosa di interessante, anche se all'inizio non è proprio chiaro in testa quello che vogliamo realizzare o dove vogliamo arrivare. Chiunque abbia mai avuto tra le mani una manciata di questi mattoncini di plastica sa che è praticamente impossibile non iniziare a giocherellarci e arrivare a mettere insieme qualche semplice modellino interessante, seppur per pura casualità. I linguaggi di programmazione attraggono in modo simile. Anche il software ha i suoi pezzi; può addirittura arrivare a costruirsi i propri: mattoncini che costruiscono mattoncini. Ci sono programmi capaci di migliorare nel tempo al fine di raggiungere un risultato in modo più efficiente di prima: apprendono. Anche se c'è una grossa differenza con il mondo delle costruzioni: il software è immateriale, non si può toccare.
Dai tempi dell'invenzione di Ada Lovelace, prima scienziata che nel 1842 diede forma a ciò che si può definire il primo software mai esistito [Men42], il mondo ha accelerato. Fare software è diventato complesso, rilevante e d'impatto. Insegnare ai calcolatori a compiere delle azioni ha portato a risvolti dirompenti, ha impresso un grande cambio di marcia all'umanità, che così veloce e collettivamente non aveva mai potuto pensare. Programmi e schede elettroniche hanno cambiato radicalmente il modo di vivere di buona parte della popolazione mondiale. Ci muoviamo molto di più e su una scala notevolmente più ampia di quanto non sia mai accaduto prima. Dall'applicazione dell'intelligenza artificiale in campo biomedico alle previsioni delle temperature del globo terrestre, ma anche alle sofisticate reti illegali che sfruttano Tor per organizzarsi e realizzare illeciti. I fasci di fibra partono dai *data-center*, arrivano alle torri LTE e finiscono via radio agli smartphone, dietro cui ci sono persone che assorbono queste informazioni. Persone che osservano, si emozionano e decidono tantissimi aspetti della propria vita.

NOTA

Tor è una delle principali reti per navigare nel *Dark Web* e nasce come progetto per preservare privacy e anonimato. Solo in un secondo momento viene sfruttato al fine di commettere illeciti. *Long Term Evolution* è lo standard di telefonia radiomobile più avanzato. A oggi largamente diffuso nella sua versione 4.5G+, è progettato per trasmettere dati con picchi di velocità di 3 Gbit/s.

La quantità di software scritto, e ancora da scrivere, aumenta di continuo. Sono lontani i tempi in cui un solo programmatore era in grado di analizzare, disegnare, testare e governare da solo l'opera di una macchina. La domanda di mercato è esplosa e così la dimensione, la quantità di codice sorgente. Dalle decine di migliaia di righe in un'app per smartphone alle centinaia di milioni di righe in un'automobile a guida autonoma. Se dovessimo stampare un milione di righe di codice sorgente su fogli A4 avremmo bisogno di almeno sedicimila fogli di carta, una pila di risme che corrisponde a circa 168 centimetri di altezza: la statura media di un uomo italiano.
Ma come inizia la vita del software? Come arriva a essere costruito e messo in esecuzione? Disponibile, sempre online per consultare, intrattenere, acquistare. Anche quando milioni di persone lo usano nello stesso istante. La cultura e le pratiche DevOps hanno l'effetto di aiutarci a imparare a gestire questi processi, che si ramificano con complessità e che trascendono il campo della sola informatica.
Chi dice che Internet e i computer abbiano cambiato gli uomini forse ha ragione, e così chi costruisce e tiene vivo il software si porta addosso la responsabilità di comprendere l'impatto delle proprie creazioni nel mondo reale: la loro qualità, il loro costo. Se maneggiate il software ogni giorno, che siate costruttori o utilizzatori, se volete iniziare da subito a capirlo meglio, il suggerimento è quello di tentare di raccontarlo e spiegarlo a chi è interessato a maneggiare questa materia. E di provarci spesso, anche le volte in cui vi sembra troppo presto, perché neanche voi avete le idee chiare. Mettere a fuoco poco alla volta i problemi complessi è naturale.

Costruire dalle fondamenta

I temi fondamentali che incrociamo in questo libro sono molteplici: benché i principali abbiano a che fare con i sistemi e la programmazione nelle organizzazioni, per capire come l'industria si sia evoluta fino al suo stato attuale è necessario introdurre qualche principio di economia, matematica, statistica, psicologia e linguistica.
Nell'informatica ci sono sempre tantissimi modi per risolvere gli stessi problemi: si dice che se avessimo dieci soluzioni potenziali per un quesito, una sarebbe sbagliata, otto corrette e una considerata (forse!) molto corretta. A tutte si accompagnano opinioni e posizioni più o meno rigide, che a volte animano discussioni anche accese.
Per navigare in questo tipo di complessità e incertezza, faremo uso di quattro fondamentali.

- L'*open source*: il tipo di software distribuito insieme con i suoi sorgenti e una garanzia sul diritto di modifica. Questo tipo di modello incentiva la collaborazione, la libera circolazione delle idee, il confronto e dà valore al ruolo della contribuzione per generare ecosistemi di conoscenza virtuosi.

- *Extreme Programming* (XP): un modo di lavorare in team che appartiene alla famiglia dei metodi per lo *sviluppo agile* di software. Affronta il mestiere di chi costruisce programmi in team e considera l'eccellenza tecnica come forza motrice per procedere rapidamente e con prudenza. Analogamente a DevOps, esplora le radici del *Lean*, modello di produzione industriale leggera che punta alla riduzione degli sprechi, all'apprendimento e al miglioramento continuo. XP contiene una parte di valori e regole astratte e una parte prescrittiva, ossia relativa a come fare o non fare qualcosa.
- La *Unix Way*: uno stile modulare di strutturazione dei nostri sistemi. Si procede per mattoncini piccoli, semplici, efficienti e riutilizzabili. Ci permette di gestire la complessità dei problemi grazie all'uso di norme e principi infrastrutturali.
- La storia di *GNU/Linux* e *Git*, il sistema operativo creato da Linus Torvalds e Richard Stallman che ha trasformato i modi di collaborare. Con il suo modello di contribuzione innovativo supportato dallo strumento di condivisione del codice *git*, l'effetto dei due progetti ha contribuito in maniera radicale alla diffusione di massa di Internet e dei suoi standard, spostando l'*open source* in campo commerciale, accademico e sociale.

Questi elementi in particolare sono considerati importanti per:

- esemplificare un metodo accessibile e senza barriere economiche;
- adottare sistemi definiti, longevi e collaudati;
- unire la vaghezza di principi e valori alla concretezza di pratiche tecniche che possono essere usate nel mondo reale.

L'esperienza di chi scrive deriva da numerose ore di lavoro spese in contesti di questo tipo. Il panorama di mercato attuale è ben più ricco di quanto non venga nominato in queste pagine e sovente coperto da estesa letteratura. Vi invito a esplorarlo e tenerlo d'occhio di frequente e in maniera attenta e critica, per capire se e quale prodotto o tecnica sia più adatto alle vostre necessità.
Una cultura aperta e di rispetto delle persone è parte fondamentale del movimento DevOps: come vuole un antico modo di dire, "sulle spalle dei giganti" per arrivare a guardare più lontano [Mer65].
Valutazioni oggettive, culturali ed economiche fanno parte di questo mestiere: osservare dove vadano investimenti e flussi di moneta (vedi il Capitolo 1) funge spesso da cartina tornasole nei trend di mercato.

Struttura del libro

Questo libro è diviso in due parti: una orientata alla teoria e al contesto, l'altra più manualistica, oltre alle appendici con le schede di approfondimento. Il tentativo è fornire un contesto teorico e i razionali sui quali sono costruiti i metodi, oltre che qualche spunto pratico per iniziare a fare piccoli passi.
Nella Parte I sono affrontati e raccontati il contesto di mercato, i fondamenti di metodi, regole, principi e valori che stanno dietro alla costruzione del software con metodi iterativi e incrementali: dall'idea di creare un programma a quando il software viene progettato e realizzato da un gruppo di persone (il team). Seguono le modalità dell'agilità di XP in chiave DevOps.

- Capitolo 1: introdurremo il contesto tecnologico del mondo moderno, i modelli di mercato in cui operano le organizzazioni, il ruolo dell'aspettativa degli utenti e della produzione di beni immateriali per un'azienda che opera con le tecnologie IT; seguirà un breve ripasso della teoria dei sistemi e la loro classificazione per complessità.
- Capitolo 2: ripercorreremo la storia del movimento DevOps dal momento della sua nascita fino ad arrivare a dare uno sguardo al possibile futuro, al fine di darne una possibile definizione.
- Capitolo 3: getteremo le fondamenta per costruire un'organizzazione DevOps attraverso un espediente, l'acronimo CALMS: Cultura, Automazione, *Lean*, Misurazione e Condivisione.
- Capitolo 4: cercheremo di capire come sia possibile intraprendere un cammino verso un cambiamento culturale che miri a DevOps, vedremo dove partire e proporremo alcuni modelli per capire che cosa succede durante il cambiamento.
- Capitolo 5: proveremo a definire il concetto di valore, capiremo in dettaglio il ruolo del *Lean* di Toyota in DevOps, al fine di organizzare un flusso di produzione senza interruzione. Completiamo la panoramica con uno sguardo alle teorie del *Product Development Flow* di Donald Reinertsen.
- Capitolo 6: da quando scrivere software è diventata un'esigenza per le aziende, sono state esplorate diverse strade rispetto ai metodi possibili per lavorare assieme. Ripercorreremo la storia dello sviluppo software dai metodi predittivi (il *Waterfall*) a quelli iterativi e incrementali (i metodi Agili), fino ad arrivare a una possibile declinazione di XP in DevOps.
- Capitolo 7: la comunicazione, durante la scrittura di software in team, ha un ruolo chiave. Cercheremo di capire quali sono i rischi più comuni nel trascurare questo elemento, analizzando poi il ruolo dei linguaggi (naturale e di programmazione) e offrendo alcune tecniche per procedere efficacemente.

Nella Parte II proveremo a costruire un ecosistema di *Continuous Delivery*: completeremo a piccoli passi (cinque esercitazioni successive) l'intera catena di rilascio, dal momento della modifica sul computer del programmatore fino all'arrivo alla produzione e alla sua messa in opera in *cloud*. Ogni capitolo contiene una parte teorica e alcuni esercizi pratici, con domande a fine capitolo per riflettere sui concetti chiave di quei passaggi che portano il codice sorgente nelle mani di chi lo adopera.

- Capitolo 8: ripercorreremo la storia delle infrastrutture software dal momento della loro invenzione all'avvento del *cloud*, sotto lo sguardo della Unix Way e di Linux. Daremo vita a una piccola *cloud* privata sul nostro computer con le applicazioni *open source*: Nomad e Consul e Fabio.
- Capitolo 9: le piattaforme software interne alle aziende sono ciò che abilita la collaborazione di un'organizzazione DevOps, per metodi e processi. Simuleremo una lavorazione su un progetto software di esempio, invieremo il contributo in forma di codice attraverso Git e lo strumento della Pull-Request come se stessimo lavorando in un team di persone.
- Capitolo 10: dal termine di una lavorazione in poi il codice lascia il computer del singolo sviluppatore e si avvia verso il cliente. DevOps punta ad automatizzare tale flusso. La prima parte di questo processo è chiamata *Continuous Integration*. Sperimenteremo che cosa significhi scrivere i test automatici, vedremo la loro funzione

di strumento a garanzia della qualità e la valenza che hanno nella progettazione. Introdurremo il concetto di pacchetto software (*build*) e costruiremo tale processo attraverso un software di automazione chiamato Jenkins.

- Capitolo 11: il design delle infrastrutture è mutato negli anni, dalle macchine virtuali ai *container*, così come da grandi programmi da migliaia di righe detti monoliti si è passati a piccoli e leggeri software adatti al *cloud* detti microservizi. Costruiremo un piccolo microservizio a partire da un esempio e lo prepareremo all'esecuzione realizzando un *container* Docker.
- Capitolo 12: dal momento in cui il software è pronto per essere introdotto nel sistema di produzione sono necessari ancora alcuni passaggi automatici: il *Continuous Delivery*. È un processo che riguarda tutto il team e gli strumenti che utilizza. Completeremo un rilascio automatico con Jenkins, realizzando tutti i passaggi necessari a far arrivare il *container* nel *cloud* virtuale messo in funzione all'inizio delle esercitazioni. Inoltre, daremo qualche spunto per organizzare un team DevOps che mantenga operativi e in salute i servizi tutti i giorni all'anno:
- Capitolo 13: ogni team che realizza e mantiene prodotti software può servirsi di uno strumento estremamente versatile per organizzare e gestire le lavorazioni, sia nel quotidiano sia nelle emergenze: la *Kanban board*. Vedremo come sia possibile realizzarne una gestendo carichi di lavoro e flussi disomogenei e imprevedibili attraverso uno strumento già noto nel mondo dell'informatica: le teorie delle code.
- Capitolo 14: la necessità di apprendere, nel mondo informatico non si esaurisce mai durante tutta la vita professionale. Faremo qualche considerazione su come apprendere più velocemente e continuare a migliorarci, senza mai fermarci ai risultati raggiunti attraverso tecniche tradizionali e il mondo dei giochi seri.

Il manuale si rivolge a una platea ampia: dallo sviluppatore alle prime armi al sistemista esperto, e in spirito Agile a coloro che non scrivono codice, come manager, analisti, coach, consulenti, imprenditori e appassionati.
Coloro che hanno familiarità con le teorie *Lean* e i principi del *Flow* nelle organizzazioni di software possono decidere di proseguire velocemente, per affrontare la parte manualistica senza proseguire necessariamente in maniera ordinata. A chi proviene da esperienze diverse dal mondo Agile, dal mondo *Scrum* o l'esperienza di singoli strumenti (come Docker o Jenkins) consigliamo, invece, di procedere con ordine.
Ai lettori più curiosi suggeriamo di dare un'occhiata alle note e ai box di approfondimento. Nel testo troverete storie, metafore ed espedienti narrativi utili a una visione generale, spesso volutamente oltre l'informatica.
Buona lettura!

Codice sorgente per esercitazioni ed esempi

I lettori che desiderino approfondire o eseguire sul proprio computer gli esercizi trovano le indicazioni per il prelievo del codice sorgente e la sua gestione nell'Appendice B, in fondo al libro. Sono inoltre disponibili ulteriori guide ed esempi. Tutti gli esempi sono prelevabili e consultabili senza registrazione attraverso un qualsiasi browser.

Suggerimenti e segnalazioni

Autore e revisori sono lieti di ricevere suggerimenti, messaggi e critiche costruttive. Abbiamo profuso ogni sforzo possibile per verificare la correttezza del testo e del materiale a corredo del libro, inclusa la menzione di fonti. Se qualcosa fosse sfuggito ai nostri occhi vi invitiamo a contattarci. È possibile inviarci un messaggio attraverso il modulo di contatto disponibile all'URL `https://devops-ief.fabiomora.com/`.

Licenze

Nei casi in cui ne sia fatta espressa menzione il testo comprende materiale tutelato da:

- licenze *Creative Commons* (anche indicate come CC), disponibili per la consultazione all'URL `https://creativecommons.org/licenses/`;
- licenze *GNU General Public License*, disponibili per la consultazione all'URL `https://www.gnu.org/licenses/licenses.it.html`.