Introduzione

Questo libro vi aiuterà a diventare programmatori migliori.
Non è importante se lavorate da soli, se fate parte di un team di progetto molto ampio o se siete consulenti che lavorano con molti clienti contemporaneamente: questo libro vi aiuterà a lavorare meglio, a livello individuale. Non è un libro teorico: ci concentriamo su argomenti pratici, su come usare l'esperienza per prendere decisioni più informate.
L'aggettivo *pragmatico* viene dal latino, *pragmaticus*, ovvero "competente nel proprio lavoro", che a sua volta deriva dal greco πραττειν, che significa "fare". Questo è un libro sul fare.
Programmare è un lavoro artigianale. Nel caso più semplice, significa convincere un computer a fare quello che volete che faccia (o che i vostri utenti vogliono che faccia). Da programmatori, siete in parte ascoltatori, in parte consiglieri, in parte interpreti e in parte dittatori. Cercate di cogliere requisiti che un po' vi sfuggono e di trovare un modo per esprimerli, così che una semplice macchina possa render loro giustizia. Cercate di documentare il vostro lavoro in modo che altri possano capirlo e cercate di ingegnerizzarlo, in modo che altri possano usarlo per costruzioni ancora più complesse. Cosa ancora più importante, provate a fare tutte queste cose mentre l'orologio del progetto continua a ticchettare senza mai fermarsi. Fate ogni giorno tanti piccoli miracoli.
È un lavoro difficile.
Molti vi offrono aiuto. Produttori di *tool* vantano le meraviglie che i loro prodotti possono fare. Guru di metodologia promettono che con le loro tecniche i risultati sono garantiti. Ognuno dice che il suo linguaggio di programmazione è il migliore, e ogni sistema operativo è la risposta a tutti i mali possibili e immaginabili.
Ovviamente, è tutto falso. Non ci sono risposte facili. Non esiste *la* soluzione migliore, che sia uno strumento, un linguaggio o un sistema operativo. Ci sono solo sistemi che possono essere più o meno adatti a un certo insieme di circostanze.
Qui entra in campo il pragmatismo. Non dovete sposare una particolare tecnologia, ma dovete avere un background e una base di esperienze sufficientemente ampi da consentirvi di scegliere buone soluzioni in situazioni particolari. Il background viene da una comprensione dei principi di fondo dell'informatica, e l'esperienza arriva da un'ampia gamma di progetti pratici. Teoria e pratica si combinano per rendervi forti.
Regolate la vostra impostazione in modo che si adatti alle circostanze e all'ambiente. Valutate l'importanza relativa di tutti i fattori che influenzano un progetto e usate la vostra esperienza per produrre soluzioni appropriate. E fate tutto questo continuamente, al procedere del lavoro. I "programmatori pragmatici" portano a termine il lavoro, e lo fanno bene.

Per chi è questo libro?

Questo libro si rivolge a persone che vogliono diventare programmatori più efficaci e più produttivi. Forse vi sentite frustrati perché non vi sembra di esprimere tutto il vostro potenziale; magari invidiate colleghi che sembra usino strumenti che li rendono più produttivi di voi; o forse state usando tecnologie datate, e volete sapere se potete applicare idee più recenti a quel che fate.
Non pretendiamo di avere tutte le risposte (e nemmeno la maggior parte), e non tutte le nostre idee sono applicabili a tutte le situazioni. Quel che possiamo dire è che, se seguite la nostra impostazione, accumulerete esperienza rapidamente, la vostra produttività aumenterà, e avrete una comprensione migliore di tutto il processo di sviluppo. E scriverete software migliore.

Che cosa fa di un programmatore un pragmatic programmer?

Ogni sviluppatore è unico, con i suoi punti di forza e le sue debolezze, le sue preferenze e le cose che non gli piacciono. Con il tempo, ciascuno si costruirà il proprio ambiente personale, che rispecchierà la sua individualità non meno dei suoi hobby, del modo di vestire o del taglio di capelli. Se siete programmatori pragmatici, avrete molte di queste caratteristiche.

- *Early adopter/fast adapter.* Avete un istinto per le tecnologie e le tecniche, e vi piace sperimentare. Quando vi imbattete in qualcosa di nuovo, lo afferrate in fretta e lo integrate al resto della vostra conoscenza. La vostra fiducia deriva dall'esperienza.
- *Curiosi*. Fate tante domande. *Bello – come hai fatto? Hai avuto problemi con quella libreria? Che cos'è quel BeOS di cui ho sentito parlare? Come sono implementati i collegamenti simbolici?* Accumulate piccoli fatti, ciascuno dei quali potrebbe influenzare qualche decisione magari a distanza di anni.
- *Pensatori critici*. Raramente date qualcosa per scontato senza conoscere a fondo i fatti. Quando i colleghi dicono "perché si fa così" o un produttore vi promette la soluzione di tutti i vostri problemi, percepite una sfida.
- *Realisti*. Cercate di capire la natura di fondo di ogni problema che incontrate. Questo realismo vi dà una buona percezione di quanto le cose siano difficili e di quanto tempo richiederanno. Capire da soli che un processo *deve* essere difficile o che *richiederà* del tempo per poter essere completato vi darà la forza di perseverare.
- *Tuttofare.* Cercate in tutti i modi di avere familiarità con molte tecnologie e molti ambienti, e cercate di rimanere aggiornati sui nuovi sviluppi. Magari la vostra attuale mansione vi costringe a essere degli specialisti, ma sarete sempre in grado di passare a nuovi campi e nuove sfide.

Abbiamo lasciato per ultime le caratteristiche più basilari, che tutti i programmatori pragmatici hanno in comune. Sono abbastanza fondamentali da formularle come suggerimenti:

SUGGERIMENTO 1

Abbiate cura del vostro mestiere.

Pensiamo che non abbia senso sviluppare software, se non si ha intenzione di farlo bene.

SUGGERIMENTO 2

Pensate al vostro lavoro.

Per essere programmatori pragmatici, dovete pensare a quello che fate mentre lo fate. Non si tratta di una valutazione *una tantum* delle pratiche correnti, ma di una valutazione critica di ogni decisione che prendete, ogni giorno, per ogni progetto. Mai mettere il pilota automatico. Dovete sempre pensare e criticare il vostro lavoro in tempo reale. Il vecchio motto aziendale della IBM, THINK!, è il mantra del programmatore pragmatico.
Se tutto questo vi sembra difficile, allora siete dei *realisti*. Ci vorrà un po' di tempo prezioso, tempo che probabilmente già scarseggia. La ricompensa però è un coinvolgimento più attivo in un lavoro che amate, una sensazione di padronanza su una serie crescente di argomenti, e il piacere di un miglioramento continuo. Sul lungo termine, l'investimento di tempo sarà ripagato: voi e la vostra squadra sarete più efficienti, scriverete codice più facile da manutenere e passerete meno tempo in riunione.

Pragmatici singoli, grandi squadre

Qualcuno pensa che non ci sia spazio per l'individualità nelle grandi squadre o nei progetti complessi. "La costruzione di software è una disciplina ingegneristica", dicono, "e tutto crolla se i singoli membri della squadra prendono decisioni per i fatti loro".
Non siamo d'accordo.
La costruzione di software *deve* essere una disciplina ingegneristica, ma questo non esclude le capacità individuali. Pensate alle grandi cattedrali costruite in Europa nel medioevo. Ciascuna ha richiesto migliaia di anni/persona di lavoro, spalmati nell'arco di decenni. Quel che veniva appreso era trasferito al gruppo successivo di costruttori, che facevano progredire con i loro risultati lo stato dell'ingegneria strutturale. Ma carpentieri, tagliatori, scalpellini e lavoratori del vetro erano tutti artigiani, che interpretavano i requisiti ingegneristici per produrre un tutto che andava al di là della pura meccanica della costruzione. Era la loro convinzione nei loro contributi individuali che sosteneva i progetti:

> Noi che tagliamo solamente pietre dobbiamo sempre avere in mente le cattedrali.
> – *Credo del cavatore*

Entro la struttura generale di un progetto c'è sempre spazio per l'individualità e le capacità artigianali, il che è particolarmente vero, visto lo stato attuale dell'ingegneria del software. Fra cento anni, forse, la nostra ingegneria sembrerà tanto arcaica quanto le tecniche degli antichi costruttori di cattedrali paiono oggi ai nostri ingegneri civili, ma le nostre capacità artigianali saranno ancora onorate.

È un processo continuo

> Un turista in visita all'Eton College, in Inghilterra, chiede al giardiniere come ha fatto a ottenere prati così perfetti. "Facile", risponde quello. "Basta spazzar via la rugiada ogni mattina, tosare l'erba un giorno sì e uno no, e rivoltare la terra una volta alla settimana".
> "Tutto qui?"; chiede il turista.
> "Assolutamente", risponde il giardiniere. "Faccia così per 500 anni e anche lei avrà un bel prato."

Bei prati hanno bisogno di poca cura quotidiana, e lo stesso vale per i grandi programmatori. I consulenti di management amano lasciar cadere nella conversazione il termine *kaizen*. È una parola giapponese che coglie l'idea di apportare continuamente tanti piccoli miglioramenti, ed è stata considerata una delle ragioni principali dei grandi incrementi di produttività e qualità della manifattura giapponese, ampiamente copiata in tutto il mondo. *Kaizen* vale anche per gli individui. Ogni giorno, cercate di perfezionare le vostre competenze e di aggiungere nuovi strumenti al vostro repertorio. A differenza dei prati di Eton, comincerete a vedere i risultati entro pochi giorni. Con gli anni, sarete stupiti da come la vostra esperienza sia fiorita e le vostre competenze siano cresciute.

Come è organizzato il libro

Questo libro è scritto sotto forma di una serie di brevi sezioni, ciascuna delle quali è indipendente e affronta un argomento particolare. Troverete molti rimandi incrociati, che vi aiuteranno a collocare ogni argomento nel suo contesto. Potete leggere le sezioni in qualsiasi ordine: questo non è un libro da leggere in fila, dalla prima pagina all'ultima. Ogni tanto incontrerete qualche riga evidenziata dal titolo *Suggerimento* (come "Abbiate cura del vostro mestiere"). Oltre a evidenziare quanto viene detto nel testo, ci sembra che questi suggerimenti abbiano una vita propria: viviamo seguendoli ogni giorno.
L'Appendice A contiene una serie di risorse: la bibliografia del libro, un elenco di URL di risorse web e un elenco di riviste, libri e organizzazioni professionali che vi consigliamo. In tutto il libro troverete rimandi alla bibliografia e all'elenco degli URL, nella forma [KP99] e [URL 18], rispettivamente.
Dove opportuno abbiamo inserito esercizi e progetti "sfida". Gli esercizi in genere hanno risposte relativamente immediate, mentre le sfide sono più aperte. Per darvi un'idea del nostro modo di pensare, nell'Appendice B trovate le nostre risposte agli esercizi, ma pochissimi hanno un'unica soluzione giusta. Le sfide possono costituire la base di discussioni di gruppo o di saggi in corsi di programmazione avanzata.

Che cosa c'è in un nome?

> "Quando io uso una parola", Humpty Dumpty disse in tono piuttosto sdegnato, "essa significa esattamente quello che voglio – né più né meno".
> – Lewis Carroll, *Alice attraverso lo specchio*

Nel corso del libro troverete qua e là un po' di gergo: parole o espressioni italiane o inglesi che sono state piegate a indicare qualche concetto tecnico, o orrende parole inventate a

cui informatici con qualche conto in sospeso con la lingua hanno assegnato un significato. La prima volta che usiamo questi termini, cerchiamo di definirli, o almeno di dare un'indicazione del loro significato. Siamo sicuri che qualcuno ci sarà sfuggito, mentre altri, come *oggetto* e *database relazionale*, sono di uso così comune che l'aggiunta di una definizione sarebbe stata noiosa. Se vi capitasse di incontrare un termine che non avete mai visto prima, non saltatelo: prendetevi il tempo per cercarlo, magari nel Web o in un manuale di informatica; e, nel caso, mandateci una email e lamentatevi, così potremo aggiungere una definizione nella prossima edizione.
Detto questo, abbiamo deciso di vendicarci degli informatici. A volte, esistono termini di gergo tecnico, che abbiamo deciso di ignorare. Perché? Perché il gergo esistente di solito è limitato a un particolare ambito di problemi, o a una particolare fase dello sviluppo, mentre una delle filosofie fondamentali di questo libro è che la maggior parte delle tecniche che consigliamo sono universali: la modularità vale per il codice, i progetti, la documentazione e l'organizzazione del team, per esempio. Usare il termine convenzionale in un contesto più ampio sarebbe fonte di confusione: non è facile trascurare il bagaglio di significati che il termine originale porta con sé. Quando ci siamo trovati in situazioni simili, abbiamo dato il nostro contributo al declino della lingua inventando i nostri nuovi termini.

Codice sorgente e altre risorse

La maggior parte del codice che troverete in queste pagine è disponibile all'indirizzo:

`https://pragprog.com/book/tpp/the-pragmatic-programmer`

Lì troverete anche collegamenti a risorse che ci sono sembrate utili, con aggiornamenti al libro e notizie di altri sviluppi da programmatori pragmatici.

Mandateci un feedback

Ci farebbe piacere sentirvi. Commenti, suggerimenti, segnalazioni di errori nel testo e di problemi negli esempi sono tutti benvenuti. Scriveteci all'indirizzo di posta elettronica `ppbook@pragmaticprogrammer.com`

Ringraziamenti

Quando abbiamo iniziato a scrivere questo libro, non avevamo idea che avrebbe finito per diventare una grande impresa di squadra.
La Addison-Wesley è stata brillante: ha preso un paio di hacker senza esperienza editoriale e li ha portati lungo tutto il processo di produzione, dall'idea al materiale pronto per la stampa. Grazie mille a John Wait e Meera Ravindiran per il loro sostegno iniziale, a Mike Hendrickson, il nostro editor entusiasta (e grande progettista di copertine), a Lorraine Ferrie e John Fuller per l'aiuto in fase di produzione e all'instancabile Julie DeBaggis per aver tenuto insieme il tutto.

Poi ci sono stati i revisori: Greg Andress, Mark Cheers, Chris Cleeland, Alistair Cockburn, Ward Cunningham, Martin Fowler, Thanh T. Giang, Robert L. Glass, Scott Henninger, Michael Hunter, Brian Kirby, John Lakos, Pete McBreen, Carey P. Morris, Jared Richardson, Kevin Ruland, Eric Starr, Eric Vought, Chris Van Wyk e Deborra Zukowski. Senza i loro commenti e le loro idee preziose questo libro sarebbe stato meno leggibile, meno preciso e lungo il doppio. Grazie a tutti per il vostro tempo e la vostra saggezza.
La seconda edizione di questo libro ha tratto grande vantaggio dagli occhi di falco dei nostri lettori. Grazie mille a Brian Blank, Paul Boal, Tom Ekberg, Brent Fulgham, Louis Paul Hebert, Henk-Jan Olde Loohuis, Alan Lund, Gareth McCaughan, Yoshiki Shibata e Volker Wurst, per aver trovato gli errori e per aver avuto la cortesia di farceli notare.
Nel corso degli anni, abbiamo lavorato con un gran numero di clienti, accumulando e perfezionando l'esperienza di cui scriviamo qui. Recentemente, abbiamo avuto la fortuna di lavorare su vari grandi progetti con Peter Gehrke: abbiamo apprezzato molto il suo sostegno e il suo entusiasmo per le nostre tecniche.
Questo libro è stato prodotto con LATEX, pic, Perl, dvips, ghostview, ispell, GNU make, CVS, Emacs, XEmacs, EGCS, GCC, Java, iContract, SmallEiffel e le shell Bash e zsh sotto Linux. La cosa incredibile è che tutto questo software è liberamente disponibile. Dobbiamo un enorme "grazie" alle migliaia di programmatori pragmatici di tutto il mondo che hanno creato queste cose per tutti noi. Vogliamo ringraziare in particolare Reto Kramer per l'aiuto che ci ha dato con iContract.
Ultimo, ma certo non per importanza, abbiamo un debito enorme on le nostre famiglie. Non solo hanno sopportato la scrittura a tarda notte, le enormi bollette telefoniche e la nostra costante aria distratta, ma hanno anche avuto la cortesia, ogni tanto, di leggere quello che avevamo scritto. Grazie per averci lasciato sognare.

Andy Hunt
Dave Thomas