LUCA MILAN

IL SENIOR AMPLIFICATO

Guidare gli agenti senza cedere il giudizio

Il senior amplificato

All’agente il completamento, al senior il giudizio.

Luca Milan

Questo libro è in vendita presso https://leanpub.com/il-senior-amplificato

Questa versione è stata pubblicata il 2026-08-12

Questo è un libro di Leanpub. Leanpub permette ad autori ed editori un processo di pubblicazione agile. La Pubblicazione Agile consite nel pubblicare un ebook in corso d’opera, utilizzando strumenti leggeri e molte iterazioni per ottenere un feedback dai lettori, al fine di assicurare un libro giusto e attraente una volta completato.

Ringraziamenti

A Liza e Federico: grazie per avermi sopportato quando la mia testa era altrove e per avermi ricordato, ogni giorno, che c'è una vita fuori dal codice.

Ancora grazie a Liza, la cui arte ha dato un volto a questo progetto con l'immagine di copertina.

Indice

Patto con il lettore

Questo libro è per te, se conosci la complessità reale dello sviluppo software e vedi mutare il tuo ruolo: non soltanto scrivere codice, ma orchestrare entità capaci di generarlo.

Oggi la sfida sta nel gestire la natura probabilistica di un agente che può offrire una soluzione brillante o una deviazione costosa; padroneggiare la sintassi e i costrutti di un linguaggio si dà ormai per acquisito. In questa transizione la competenza Senior si sposta dalla scrittura diretta alla progettazione dei sistemi che governano il lavoro dell'agente.

Per dare forma a questa relazione, il libro usa il termine **System Harness** per l'infrastruttura che avvolge l'agente, ne delimita le mosse e conserva le decisioni insieme alle ragioni che le hanno prodotte. Il perimetro preciso arriva nell'Introduzione; qui basta il nome.

Quello che il libro racconta è un esperimento tecnico: costruire con un agente un rapporto di lavoro che regga nel tempo. Un rapporto così non si dichiara in un prompt di avvio, e regge soltanto se l'agente non riparte da zero a ogni sessione. Disegno, limiti e condizione di smentita stanno in *L'esperimento*, poco più avanti.

Questo libro non promette produttività automatica, non è un manuale d'uso e non mette a disposizione un repository di codice da scaricare. Quello che offre è **ingegneria del rapporto collaborativo**: i confini, le responsabilità e le logiche con cui un Senior trasforma la generazione di testo in lavoro governato.

Una precisazione, perché l'equivoco è frequente: questo non è un libro contro gli strumenti, né a favore di uno in particolare. Quale assistente si usi conta molto meno di dove passa il confine della delega, e gli strumenti cambiano più in fretta di quanto un libro si scriva.

> **Il problema non è usare Claude Code, Cursor o Codex. Il problema è delegare il giudizio invece dell'implementazione**.

La delega del giudizio non ha un interruttore, e quasi nessuno la dichiara: avviene per gradi, mentre il lavoro continua a sembrare in ordine. Per questo il libro insiste sui confini, sulle verifiche e sulle condizioni di arresto invece che sulla scelta dello strumento.

Gli esempi del libro sono fittizi: personaggi e situazioni inventati a partire da esperienze di mestiere, per isolare un principio senza il rumore del caso originale. L'harness che racconto fa eccezione: ne descrivo capacità e scelte verificate sulla sua implementazione.

Resta **la responsabilità dell'errore**. Il Senior delega l'esecuzione e alcune decisioni operative circoscritte, ma non trasferisce all'agente né al System Harness la responsabilità finale delle conseguenze: la esercita attraverso l'harness, che rende espliciti confini, verifiche e condizioni di arresto. Che si usi l'harness descritto in questo libro o un altro, del risultato risponde sempre una persona: il Senior. Nel tuo progetto, quella persona sei tu.

Apertura

Prima dei capitoli, quattro testi.

- **La nota dell'autore** dice perché il libro nasce.
- **L'introduzione** fissa il problema da cui tutto parte: che cosa deve sopravvivere a una sessione di lavoro con un agente.
- **L'esperimento** espone il disegno con cui provo a rispondere, i suoi limiti e la condizione che lo smentirebbe.
- **I fondamenti** separano i tre livelli — principio, metodo e implementazione — che i capitoli daranno per acquisiti.

Perché questo libro

Vent'anni di esperienza sul campo, passati a scrivere codice e a rilasciare servizi e prodotti digitali, mi hanno insegnato che non c'è nulla di più stimolante che risolvere problemi complessi e costruire strumenti che semplificano la vita ad aziende e utenti. Ora quel mestiere sta cambiando, e questo libro nasce per capire in che cosa.

Da quando lavoro con un agente, scrivere codice mi occupa meno tempo di prima, e cresce tutto ciò che sta attorno: decidere che cosa costruire, stabilire fin dove l'agente può spingersi, controllare quello che torna indietro dalle iterazioni. Finché questa capacità di discernimento e direzione resta mia, il mestiere non si impoverisce. Il rischio è cederla senza accorgermene.

Contro questo rischio sto costruendo quello che il libro racconta: un metodo, un *harness* e un progetto vero su cui collaudare l'uno e l'altro.

Il libro ha due parti.

La prima riguarda il mestiere: che cosa resta del lavoro di un Senior quando scrivere righe di codice non è più il collo di bottiglia, che differenza c'è fra esperienza e discernimento, come si formula un mandato, come si verifica un lavoro che non si è svolto in prima persona, come sopravvive alla sessione ciò che Senior e agente hanno compreso insieme. Questa parte non dipende da un framework agentico in particolare: i principi valgono con l'assistente di oggi e, spero, con quello che lo sostituirà.

La seconda parte scende dove quei principi si materializzano in comportamento: i punti di un framework agentico su cui un Senior può mettere le mani — il testo che vincola l'agente, le procedure che l'agente può richiamare, gli strumenti di cui dispone, il ciclo di vita della sessione, gli hook di Git. Racconto le scelte che ho fatto, comprese quelle che non hanno retto, perché il metro non è che il lettore capisca il mio harness: è che sappia dove intervenire nel proprio.

Chi è il Senior amplificato

Il soggetto di questo libro è una persona, non uno strumento: il Senior — chi conosce il lavoro, decide e risponde delle conseguenze — nel momento in cui l'intelligenza artificiale ne estende la portata senza sostituirne il giudizio. «Amplificato» dice questo: non un

Senior rimpiazzato dall'agente, ma un Senior che arriva più lontano restando sé stesso, con l'esperienza, i vincoli e i pattern che ha maturato negli anni.

In concreto, amplificare vuol dire non lasciare che l'agente improvvisi: trasformare quel patrimonio in vincoli riusabili e conservarli in modo che l'agente li ritrovi e li applichi.

Il metodo e l'harness con cui metto alla prova la mia tesi sono SAF, *Senior Amplification Framework*. Lo dico subito: SAF è un banco di prova, non il protagonista del libro; è l'esperimento con cui verifico come l'amplificazione avvenga davvero.

Oltre le istruzioni: la cabina di pilotaggio

Quei vincoli non si costruiscono scrivendo semplici prompt. Credo che il settore si sia concentrato troppo sulla parte facile: le istruzioni nel contesto del modello, dalla singola richiesta alle regole di comportamento. Servono, ma un'istruzione orienta la risposta, non la determina: i modelli producono output probabilistici, e la stessa istruzione può portare a esiti diversi da una sessione all'altra.

Quando l'agente tocca il lavoro vero, lo scarto tra istruzione ed esito diventa il problema.

L'approccio serio comincia dove le istruzioni finiscono: nella personalizzazione del framework agentico — come costruisce il contesto, quali eventi espone, dove un hook rifiuta un'azione. Questa è la cabina di pilotaggio: chi scrive gli strumenti decide che cosa l'agente può fare, su quale ambiente e con quale traccia di esecuzione.

Un'istruzione chiede al modello di evitare un'azione; uno strumento che non esiste gliela rende impossibile lungo quel percorso.

In tutto questo non c'è niente di esotico: è il mestiere di sempre dello sviluppatore, che si confronta con API, SDK e punti di estensione. Chi costruiva servizi con le proprie mani ora costruisce i vincoli attorno all'agente. Il libro segue questo spostamento: dalle istruzioni agli strumenti e ai vincoli che delimitano le mosse dell'agente. Il primo passo, però, non riguarda ancora gli strumenti: riguarda ciò che a una sessione di lavoro non sopravvive.

Introduzione: che cosa deve sopravvivere a una sessione

Una sessione di lavoro con un agente ha un difetto semplice: finisce.

Dentro quella sessione può succedere molto. Magari stai costruendo un backend, una UI, un servizio. L'agente propone una struttura; tu gli fai notare un vincolo che non poteva conoscere. L'agente trova un'alternativa; tu riconosci un errore che hai già visto mille volte. Insieme cambiate direzione, scartate una soluzione plausibile, stabilite che cosa non deve rompersi e decidete quale evidenza renderà credibile l'output prodotto.

Alla fine resta l'artefatto: codice, test, validazione, configurazione, documentazione. Probabilmente resta anche la conversazione. Ma ciò che ha reso buona quella sessione rischia di non rimanere affatto: perché una proposta è stata rifiutata, quale precedente ha cambiato la decisione, quale eccezione vale soltanto in questo sistema e per quel cliente, in quale punto l'agente deve fermarsi la prossima volta.

Nella sessione successiva, l'agente può avere davanti tutto il repository e non sapere niente di tutto questo. Può rileggere i file e disporre della conversazione, ma non per questo sa che cosa il Senior e l'agente hanno imparato, né conosce la strada percorsa per arrivare a una scelta.

> **Il costo non è soltanto il tempo necessario a rispiegare. È molto di più. È il rischio di decidere diversamente, perché la ragione della decisione già presa è scomparsa.**

Il libro comincia da quella perdita di informazioni.

Che cosa si ha davanti

Prima di chiedersi che cosa un Senior possa delegare, conviene guardare la natura del sistema con cui si ha a che fare.

Un **modello linguistico** produce risposte probabilistiche a partire dal contesto disponibile e dai dati con cui è stato addestrato. Da solo non apre un file, non esegue un comando e non conserva nulla della sessione precedente.

Un **agente** nasce quando un **framework agentico** — il software che circonda il modello e lo mette al lavoro — lo colloca dentro un ciclo. L'accezione è operativa: non la libreria con cui uno sviluppatore si costruisce un agente su misura, ma il software che si adotta già costruito e si configura nei punti che espone. Il framework assegna al modello gli strumenti, gli permette di osservare l'ambiente, interpreta le richieste del modello, esegue le azioni e al passo successivo gli restituisce gli effetti. La parte del framework che esegue quel ciclo è il **runtime**. Il comportamento del modello dipende quindi anche dal framework che lo ospita: dal modo in cui costruisce il contesto, espone gli strumenti, conserva lo stato, gestisce i permessi, intercetta gli eventi e decide quando il ciclo termina.

Questa composizione rende l'agente molto potente ma anche discontinuo nell'operatività di lungo periodo. La finestra corrente può contenere una discussione molto ricca e perderne il significato alla sessione successiva. C'è di più: al crescere della finestra le prestazioni tendono a peggiorare, sia perché l'attenzione del modello si distribuisce peggio su un contesto lungo, sia perché nel frattempo il contesto accumula informazione superata o contraddittoria. Il fenomeno va sotto il nome di **context rot**.[1]

Il framework può conservare la chat o la cronologia della conversazione. Questa memoria non strutturata, però, non distingue una decisione da un'ipotesi: conserva il risultato senza rendere espliciti l'alternativa scartata, il vincolo che ha determinato la decisione o la condizione che obbligherebbe a riaprirla.

Se poi il Senior cambia framework, cambiano i formati, gli eventi, gli strumenti e i meccanismi di memoria: ciò che era stato appreso rischia di restare intrappolato nell'ambiente che lo ha prodotto.

Un Senior può governare il sistema che si trova davanti? Su questo interrogativo poggia il resto del libro, ma prima di affrontarlo conviene spostare lo sguardo altrove, perché il lavoro vero non si gioca dove sembra.

Ciò che conta è il rapporto, non la risposta

Quello che fa la differenza è ciò che accade tra una risposta e l'altra. Il Senior porta esperienza puntuale, responsabilità e conoscenza del sistema; l'agente porta capacità di esplorazione, esecuzione e confronto. Correzioni, obiezioni, decisioni e verifiche danno forma al lavoro. Nel confronto, l'esperienza smette di essere soltanto un ricordo personale e diventa un criterio applicato al progetto.

[1] Il nome viene da Chroma Research: Kelly Hong, Anton Troynikov, Jeff Huber, «Context Rot: How Increasing Input Tokens Impacts LLM Performance», 14 luglio 2025, il rapporto che ha battezzato e misurato il fenomeno su diciotto modelli. La distinzione fra il degrado dovuto alla lunghezza della finestra e quello dovuto al contenuto accumulato è di Jake Minns, «Context Rot: Why Claude Code Sessions Decay, and How to Govern Them», Towards Data Science, 13 luglio 2026, che raccoglie gli studi sperimentali sul primo.

In questo libro, il termine **giudizio** indica la capacità, radicata nel contesto, di scegliere in condizioni di informazione incompleta, riconoscere i vincoli strutturali, valutare i compromessi e farsi carico delle conseguenze. L'agente prende già decisioni operative: esplora alternative, ordina i passaggi e sceglie implementazioni entro perimetri definiti. Il giudizio del Senior, però, non si trasferisce in blocco: la responsabilità finale e le decisioni direzionali restano in capo a lui.

Una parte di questo giudizio produce esiti che possono essere resi espliciti: una decisione motivata, un vincolo strutturale da presidiare, una lezione ricavata da un errore, un dubbio da mantenere vivo o una condizione che obbliga a fermarsi.

> **Questi esiti sono «distillabili»: dal flusso informale del confronto fra Senior e agente si può estrarre la sola parte che ha valore oltre la sessione**.

L'obiettivo non è trasferire il giudizio all'agente, ma condensarne le conclusioni — decisioni, vincoli, lezioni — in una forma che l'agente possa consultare nelle iterazioni successive.

La distillazione non ricade però sul Senior come lavoro manuale di fine giornata. È il **System Harness** a sostenerla (con quale perimetro si vede più sotto): guida l'agente nell'estrarre gli esiti dell'iterazione, dà loro una forma e conserva le tracce oltre la sessione corrente. Il Senior sceglie quali iterazioni valga la pena distillare e quali esiti rendere persistenti; il sistema fa in modo che ciò che passa di lì prenda una forma riutilizzabile.

La distillazione serve a impedire che il giudizio già esercitato evapori alla chiusura della finestra. Che poi quelle tracce tornino davvero utili nell'iterazione successiva è l'ipotesi che il libro mette alla prova, non un risultato acquisito: una traccia recuperata resta contesto da valutare, non un ordine da eseguire. Che una traccia sia vicina per significato non la rende pertinente, e una decisione valida ieri può essere stata superata.

Memoria e cognizione non sono la stessa cosa

Un archivio di conversazioni è memoria. Conserva ciò che è stato detto. Anche un repository è memoria: conserva gli artefatti che il lavoro ha prodotto.

La memoria è necessaria, ma non basta. Si può conservare tutto e non sapere che cosa conta, o ritrovare due decisioni incompatibili senza sapere quale delle due sia ancora in vigore.

In questo libro, **cognizione persistente** indica una pretesa più precisa: conservare decisioni, criteri, lezioni, dubbi e collegamenti in modo che il sistema possa recuperare che

cosa vale, perché vale, da dove arriva, che cosa ha superato e quando deve essere rimesso in discussione.

Questa pretesa non significa attribuire una mente al repository. «Cognizione» è il nome dell'ipotesi operativa che il libro mette alla prova: una parte del sapere maturato nella collaborazione può essere distillata in schede — i documenti in cui una decisione viene scritta insieme alle sue ragioni — collegate con sufficiente precisione da tornare pertinenti e utilizzabili in una sessione futura?

La differenza è nel verbo. La memoria **conserva**. La cognizione persistente deve anche permettere di **orientarsi**.

I collegamenti cambiano la natura dell'archivio. Una scheda isolata ricorda una conclusione. Una scheda collegata può dire quale fonte l'ha informata, quale iterazione l'ha prodotta, quale scelta precedente estende, quale vincolo la rende corrente e quale decisione successiva l'ha superata. Il valore aggiunto non è avere più testo: è poter ricostruire, a partire dal risultato, la strada che lo ha giustificato.

Torna alla sessione dell'inizio. Il vincolo che avevi fatto notare all'agente, l'alternativa scartata, la ragione per cui una soluzione plausibile non andava bene: separati restano appunti. Collegati sono il percorso che ha portato fin lì, e un'altra sessione può ripercorrerlo.

Una parola già in circolazione

Il termine *harness* circola già. Trivedy chiama *agent harness* l'insieme di codice, configurazione e logica di esecuzione che circonda il modello, ne gestisce lo stato, esegue gli strumenti e applica vincoli;[2] la documentazione di Claude Code descrive la stessa separazione dall'altro lato, dove i modelli ragionano, gli strumenti agiscono e l'harness fornisce contesto ed esecuzione.[3]

Il System Harness di questo libro riprende il termine con un perimetro diverso, e le due funzioni annunciate nel Patto con il lettore ricevono qui il loro nome. Delimitare le mosse dell'agente significa governare il **mandato**, il contratto con cui il Senior circoscrive ciò che l'agente può fare: la prima parte formula quel contratto per intero. Conservare le decisioni insieme alle loro ragioni significa sostenere la persistenza degli esiti del giudizio, e la cognizione persistente è una delle proprietà richieste al sistema. Come il System Harness faccia l'una e l'altra cosa è materia della seconda parte.

[2] Vivek Trivedy, «The Anatomy of an Agent Harness», LangChain Blog, 10 marzo 2026.
[3] Anthropic, «How Claude Code works», documentazione di Claude Code, consultata il 20 luglio 2026.

Da dove parte il libro

La prima parte comincia dalla frattura che sposta il baricentro del mestiere dello sviluppatore dalla produzione di codice alla decisione su che cosa costruire e a quali condizioni. Da lì scompone il lavoro del Senior nelle condizioni che lo rendono possibile, una per capitolo, e alla fine lo ricompone nella figura del Senior amplificato.

Il libro però non parte da ciò che l'agente sa fare in una risposta: parte da ciò che il Senior e l'agente riescono a non perdere tra una risposta e la successiva.

Con quale disegno ho provato a non perdere le decisioni e le loro ragioni è il capitolo che segue.

L’esperimento

Gestisco spesso più progetti contemporaneamente, e in ciascuno tengo insieme fronti diversi: architettura e servizi, interfaccia e infrastruttura, il codice da scrivere e i rilasci da preparare, ciò che gira in produzione e le decisioni di prodotto. La difficoltà che accomuna questi fronti è rientrare nel progetto: ritrovare, ogni volta, non solo dove mi trovavo ma perché avevo deciso così.

Da lì nasce l’impostazione operativa di questo libro. Rientrare in un progetto non è un lavoro di lettura: è una conversazione con qualcuno che quel progetto lo conosce già. Per questo motivo gestisco l’agente non come un completatore al quale impartire ordini sempre più lunghi, né come un collega immaginario al quale delegare responsabilità, ma come un **pair**: una controparte con cui esplorare, contestare, verificare e decidere.

Perché questa relazione funzioni, gli esiti che il Senior sceglie di rendere persistenti devono lasciare tracce strutturate, coerenti e consultabili nel tempo. Da qui l’obiettivo che mi sono prefissato:

> **Costruire attorno all’agente un System Harness che dia forma agli esiti del giudizio scelti per la persistenza, così che ogni decisione conservi le ragioni che l’hanno prodotta. Poi verificarne l’effetto: capire se, dopo molte sessioni, le decisioni e le loro ragioni si ritrovano, anziché dover essere ricostruite a partire dal codice o dalle conversazioni passate.**

Il disegno

Il percorso dell’esperimento si articola in tre passaggi:

1. **Osservazione**: analizzo i modelli linguistici dentro i framework agentici, per capire dove accelerano, dove inventano continuità, quando seguono una regola e quando la perdono.
2. **Interazione**: studio il modo in cui un Senior guida il modello: formulare un mandato, concedere decisioni operative, trattenere quelle direzionali, progettare verifiche e arresti.
3. **Progettazione**: traduco queste lezioni nella costruzione di un **System Harness** personalizzato.

I limiti

Non sto cercando di dimostrare che sia stata **«salvata la cognizione»** in senso forte, né che una memoria collegata renda sicuro un agente o più produttivo un team. L’obiettivo è più circoscritto. Dopo molte sessioni, il progetto riesce a dire perché un simbolo nel codice dipende ancora da una decisione ormai superata? Una correzione già fatta modifica davvero il lavoro successivo? Una lezione resta corrente finché resta corrente la causa che l’ha prodotta?

Fisso adesso due regole, prima di sapere come andrà.

La prima riguarda ciò che affermo: ogni proprietà che attribuisco al System Harness deve avere un controllo che si può eseguire. Se non so indicare come si verifica, non ho descritto il sistema: ho descritto quello che speravo.

La seconda riguarda quando smettere di difenderlo, ed è la **condizione di smentita**: se una decisione che il Senior aveva scelto di rendere persistente deve essere ricostruita a partire dal codice, invece di essere ritrovata, il disegno ha fallito nel punto in cui prometteva di più.

Cinque domande da portare nel tuo contesto

Ti propongo di portare questo esperimento nei tuoi progetti, attraverso cinque domande:

- quale contesto rendere disponibile;
- che cosa delegare;
- quale evidenza pretendere;
- quando fermare l’agente;
- quali esiti del giudizio rendere persistenti.

Queste domande funzionano soltanto se le rendi esplicite, se le metti alla prova durante le iterazioni che sostieni con l’agente e se le correggi là dove la collaborazione tra voi degrada. Ognuna, però, si può porre a tre livelli diversi: ciò che deve restare vero, il modo in cui si valuta il caso presente, la forma tecnica scelta. Confondere questi livelli è il modo più rapido di rispondere correttamente alla domanda sbagliata — ed è la confusione che il capitolo seguente prova a togliere di mezzo.

Fondamenti: tre livelli da non confondere

Un esempio ipotetico, piccolo.

Stai lavorando con un agente al modulo di fatturazione di un ERP proprietario. Per risolvere un caso particolare, l'agente propone di aggiungere un parametro booleano al costruttore di `TaxCalculator`. Sarebbe il decimo parametro, l'ennesimo inserito dopo diverse variazioni dei requisiti da parte del cliente.

La modifica funziona. Il codice compila, il test è verde e il problema immediato scompare. Eppure qualcosa non torna. Rileggi la classe e intuisci che quel nuovo flag rende ancora più difficile l'uso della classe e apre la strada ad altre eccezioni dello stesso tipo.

L'agente sta risolvendo il caso che ha davanti. Tu riconosci anche il problema che quella soluzione creerà, forse non subito, ma a distanza di qualche settimana. Perciò lo fermi e ne discutete: cambio di rotta, niente nuovo flag; le opzioni variabili vanno raccolte in un oggetto separato, e così, senza troppa fantasia nel nome, nasce la classe `TaxCalculatorOptions`.

Questa scelta coinvolge i tre livelli annunciati alla fine di *L'esperimento*: **principio**, **metodo** e **implementazione**. Distinguerli è importante, perché non invecchiano alla stessa velocità e non si delegano nello stesso modo.

Il principio: ciò che deve restare vero

Il principio che orienta la scelta è semplice: ciò che è essenziale per un oggetto non va confuso con le opzioni che cambiano da caso a caso. Quando tutto finisce nella stessa firma, la complessità non scompare. Diventa soltanto più difficile da vedere.

Questo principio non dipende dalla classe `TaxCalculator`, dal linguaggio usato o dal nome scelto per la soluzione. Può restare valido anche quando cambiano il progetto e gli strumenti.

Un **principio** indica quindi ciò che si vuole preservare. Non dice ancora quali passi compiere né quale forma debba avere il codice.

Il metodo: come orientare la scelta

Per applicare il principio, distingui nella firma le dipendenze necessarie dalle impostazioni facoltative. Poi serve verificare se il problema è isolato o se le eccezioni si stanno accumulando, nella stessa firma o in altre classi. Nel primo caso, un parametro dedicato può restare un compromesso locale; nel secondo, il nuovo flag prolunga un pattern di eccezioni. È questo passaggio a cambiare l'esito: la stessa analisi può portare a mantenere il parametro, raccogliere le opzioni o intervenire su una struttura più ampia.

Una terza verifica non si legge nel codice: chiedere se su quella firma pesa già una decisione, e a quali condizioni. Una scelta presa per questo cliente e per un periodo definito non lascia segno nella firma né nei requisiti: se non torna sul tavolo, l'analisi resta corretta e l'esito cambia.

Questo è il **metodo**: osservare il sistema, interrogare ciò che è già stato deciso e scegliere senza ridurre l'analisi alla regola «molti parametri, sempre un oggetto di opzioni». Rende il principio utilizzabile, ma non decide al posto tuo.

L'implementazione: la forma scelta adesso

In questo caso la scelta è introdurre la classe `TaxCalculatorOptions`: un oggetto che raccoglie le opzioni e lascia nel costruttore soltanto ciò che è necessario.

Questa è l'**implementazione**. È la forma tecnica adottata per lo specifico caso d'uso. In un altro linguaggio potrebbe essere un record, una configurazione o un costruttore dedicato. Domani una soluzione migliore potrebbe sostituirla senza rendere falso il principio e senza cancellare il metodo usato per arrivarci.

La differenza può essere riassunta così:

- il **principio** chiarisce che cosa va preservato;
- il **metodo** guida il modo in cui si valuta il problema;
- l'**implementazione** realizza la scelta con gli strumenti disponibili.

Perché questa distinzione conta con un agente

Un agente lavora sempre attraverso un'implementazione concreta: modifica quel file, usa quella libreria, esegue quello strumento. Può anche confrontare alternative e prendere decisioni operative entro il mandato ricevuto.

Il Senior, però, riconosce quale principio è in gioco e quali scelte richiedono una decisione direzionale: non perché l'agente sia incapace di proporre una buona soluzione, ma perché il livello in cui l'agente si muove non è quello in cui la scelta si giudica.

Confondere i tre livelli produce due errori opposti. Il primo consiste nel trattare un'implementazione come una verità permanente: «per ogni costruttore di ogni classe si usa sempre un oggetto di opzioni». Il secondo lascia il principio troppo vago: «mantieni il codice pulito». Nel primo caso lo strumento diventa una regola. Nel secondo la regola non aiuta a decidere.

Una buona collaborazione rende espliciti tutti e tre i livelli: che cosa deve restare vero, come valutare il caso e quale soluzione adottare in quel momento.

Il giudizio deve sopravvivere alla scelta

Tra un mese, in una nuova sessione, l'agente vede `TaxCalculatorOptions` e basta. Dal codice ricava che le opzioni sono state separate; non ricava perché. È la domanda che, un mese dopo, faresti davvero:

> **«Il cliente chiede l'ennesima variante fiscale. La aggiungo a `TaxCalculatorOptions`?»**

L'agente risponde. Risponde anche bene: sì, è il posto previsto, la classe esiste per quello. La risposta è plausibile, coerente con tutto ciò che ha davanti, e ti riporta dentro l'accumulo di eccezioni che avevi fermato — perché quel giorno, davanti al flag, non avevi deciso «raccogliere le opzioni»: avevi deciso di smettere di trattare come opzione ciò che cambia il calcolo. Un agente senza tracce non risponde «non lo so»: risponde, e ti aiuta volentieri a rifare la strada che avevi chiuso.

Quando la decisione è registrata, invece, quella domanda ne apre altre, e sono tutte domande a cui si può rispondere:

- un flag qui è già stato proposto, e rifiutato?
- per quale ragione — e quella ragione vale ancora oggi?
- che cosa la decisione ha toccato, e che cosa ha lasciato deliberatamente fuori?
- quale condizione la riaprirebbe?
- quale alternativa era sul tavolo, e perché è stata scartata?
- esiste una decisione successiva che l'ha superata?

Nessuna di queste risposte sta nel codice.[1] Qualcuna potrebbe stare nella conversazione di un mese prima, se non è andata persa: ma starci non basta, perché nessuno sa più in quale, e niente dice se quella decisione sia ancora in vigore.

Tre di quelle domande — che cosa la decisione ha toccato, quale alternativa è stata scartata, quale decisione successiva l'ha superata — non si risolvono registrando: si risolvono **collegando**, come ha già mostrato l'Introduzione. Ne segue una regola di scrittura: una scheda si scrive già collegata, perché formalizzare significa registrare la decisione e dichiarare a che cosa si lega, nello stesso gesto.

Perciò, chiusa la discussione, Senior e agente fanno un passo che il codice non richiede: rileggono ciò che si sono detti ed estraggono la parte che vale oltre il caso presente — il problema incontrato, le alternative confrontate, la decisione e le condizioni che potrebbero riaprirla. È l'operazione che l'Introduzione chiama distillazione, e non è un gesto a mano libera: il System Harness ne detta la forma. Il Senior decide quali esiti meritino di essere distillati e risponde di ciò che resta scritto; a scriverlo è l'agente, che quella discussione l'ha appena attraversata. Il risultato ha una forma, e quella forma è la **scheda**.

Il prompt orienta, la scheda ricorda

A prima vista, davanti a un problema ricorrente, verrebbe da affidarsi a una soluzione nativa del framework agentico in uso: aggiungere una riga nel system prompt («quando un costruttore cresce troppo, separa le opzioni») oppure scrivere una skill che guidi l'agente a riconoscere il problema e ad applicare il pattern previsto.

Entrambe le strade sono utili, ma risolvono un problema diverso. Il prompt orienta il comportamento; la skill rende ripetibile una procedura generale. Né il prompt né la skill conoscono la decisione presa in questo progetto e in questo momento: valgono ovunque, ed è proprio questa universalità a renderli sordi quando la domanda riguarda il perimetro esatto di `TaxCalculator`.

Il vero banco di prova è il vincolo che nasce dentro il rapporto di lavoro tra Senior e agente: un regime fiscale tenuto fuori dal calcolo perché vale soltanto per quel cliente e soltanto fino alla chiusura dell'esercizio. Nessun requisito di sistema lo documenta e nessuna regola generale lo prevede; il Senior e l'agente lo hanno stabilito discutendo un caso reale, e fuori da quella discussione quel vincolo non esiste. Se la decisione non viene registrata

[1] Peter Naur, «Programming as Theory Building», *Microprocessing and Microprogramming* 15, n. 5 (maggio 1985), pp. 253–261; letto il 5 agosto 2026 in trascrizione digitale, non sull'originale a stampa. Naur sostiene che la parte essenziale di un programma — la teoria di come certe faccende del mondo vengono trattate dal programma — vive in chi lo ha costruito, e che il testo del programma insieme alla sua documentazione si è rivelato «insufficient as a carrier of some of the most important design ideas». La tesi riguarda squadre di persone che si sciolgono; portarla su un agente che riparte da zero a ogni sessione è mio.

e collegata al punto del codice su cui incide, alla sessione successiva l'agente la ignorerà: tratterà quel regime come un'eccezione da correggere, oppure non vedrà affatto il perimetro in cui il vincolo vive.

È di nuovo la distinzione dei tre livelli, applicata questa volta a ciò che si decide di conservare:

- Un principio va conservato perché resta vero ovunque.
- Una decisione va conservata insieme alle condizioni che la tengono in vita, altrimenti la sessione successiva la interpreterà come una regola astratta, da applicare ovunque.

Distinguere i livelli serve a questo: sapere che cosa merita di restare, e con quali confini.

La scheda porta anche un nome più preciso: **traccia cognitiva**, un termine che sposta l'attenzione dal formato del documento alla sostanza di ciò che vi è custodito. La scheda non è il log o il riassunto della conversazione. È il **verbale strutturato del giudizio che la collaborazione ha reso esplicito e che il Senior ha validato**. Una scheda reale, con la lettura di ciò che vi si vede, è riportata in appendice.

Da qui derivano i due elementi del System Harness che rendono operativa questa separazione, e che riprenderò più avanti:

1. Il **binario di scrittura**: detta la forma della scheda e guida l'agente che la scrive, così che la decisione conservi il problema da cui nasce, i collegamenti e le condizioni che potrebbero riaprirla; non rende la decisione corretta, la rende leggibile, contestabile e coerente con le altre.
2. L'**accesso semantico**: cerca per significato le tracce che potrebbero servire. Davanti a un problema simile, l'agente non deve ricordare le parole esatte di mesi prima: può recuperare un principio generale o una decisione specifica insieme alle rispettive ragioni e ai rispettivi confini.

Il principio resta. Il metodo permette di applicarlo. L'implementazione cambia. Il giudizio tiene insieme i tre livelli, risponde delle conseguenze e, quando la scelta lo merita, lascia una traccia persistente. Da questa distinzione parte il resto del libro, esattamente là dove finisce il metro con cui uno sviluppatore misurava il proprio contributo — scrivere codice — e comincia il lavoro di governo.

Parte 1

I capitoli che seguono costruiscono una componente alla volta il **modello operativo**: l'organizzazione esplicita di ruoli, responsabilità, verifiche e apprendimento con cui Senior e agente lavorano insieme.

- **La frattura** mostra che cosa resta del mestiere quando produrre codice non è più il fattore limitante: la decisione situata, non l'esecuzione.
- **Il giudizio** dà un nome a quella capacità e la distingue dall'esperienza.
- **La cooperazione** riconosce che il rapporto con l'agente va progettato, e annuncia le parti del modello operativo.
- **Il mandato**, **l'autonomia** e **l'arresto** fissano il contratto operativo fra Senior e agente.
- **La verifica** affida la garanzia a un oracolo indipendente, deciso prima del lavoro.
- **La persistenza** e **il contesto** trasformano il giudizio in cognizione che sopravvive alla sessione.

La frattura

Immagina Tommaso e Sandro, due Senior davanti allo stesso ticket Jira: «Aggiungere l'esportazione in CSV alla pagina dei report». Nella mia esperienza, questo passa già per un buon ticket. Hanno a disposizione lo stesso agente e un'ora di lavoro.

Tommaso legge e parte. Descrive il compito all'agente, ottiene una funzione pulita, la collega al bottone e verifica che il file venga scaricato. In cinquanta minuti la funzionalità è pronta. Risponde alla richiesta e chiude il ticket.

Sandro, prima di scrivere qualsiasi cosa, si ferma su una parola: *report*. «Quali report?», si chiede. Verifica chi userà l'esportazione, con quale frequenza e dentro quale processo. Scopre che il CSV non è il reale bisogno della richiesta, bensì una soluzione che il ticket contiene già. Gli utenti vogliono riconciliare i numeri con un altro sistema di rendicontazione, e l'esportazione trasferirebbe su di loro un passaggio manuale quotidiano. Si aprono così domande che il ticket iniziale non conteneva: chi possiede il dato, quale formato accetta il sistema di rendicontazione, come vengono gestiti errori e duplicati, chi risponde quando la riconciliazione non ha successo. Prima di costruire, Sandro riapre la richiesta: forse serve una vista, forse un'integrazione, forse il CSV resta il compromesso meno costoso. Non ha ancora tutti gli elementi per scegliere l'implementazione corretta.

Tommaso ha prodotto codice conforme alla lettera del ticket prima di chiarire il problema. Sandro, per quasi tutta l'ora, non ha prodotto codice, ma ha evitato che **una soluzione non ancora giustificata diventasse subito software da mantenere**.[1]

Questa scelta ha un costo, per l'azienda e per Sandro. Il ticket resta aperto, bisogna coinvolgere gli utenti e la consegna slitta. Dopo il confronto con loro si potrebbe perfino decidere di rilasciare comunque il CSV, ma come soluzione ponte, accettandone il lavoro manuale e fissando il momento in cui rivederla. Il costo resta, ma non è più nascosto dentro una richiesta apparentemente già risolta.

Sandro non ha scritto codice migliore. Ha riconosciuto che, in quel momento, il problema non era eseguire: bisognava prima decidere che cosa valesse la pena costruire. Il codice, lì, non era il fattore limitante.

[1] Costruire una soluzione non ancora giustificata che poi andrà mantenuta alimenta ciò che in inglese si chiama *product debt*, per analogia con il *technical debt*: il costo futuro di aver costruito la cosa sbagliata, quando il codice diventa una passività da testare, aggiornare e infine rimuovere.

Il fattore limitante si è spostato

Per molto tempo, tradurre una decisione in codice richiedeva competenza e assorbiva gran parte della giornata di uno sviluppatore. Era un passaggio obbligatorio, e la velocità con cui lo si faceva era facile da osservare e misurare. Era naturale finire per usare quella capacità come indice del contributo professionale, e spesso era anche ciò per cui si veniva pagati.

Oggi, in una parte crescente di quel lavoro, **ottenere codice *plausibile* richiede molto meno tempo**. Questo guadagno **non rende facile produrre codice corretto**, **integrato**, **sicuro e mantenibile**; nei domini critici, nel legacy intricato e nei problemi algoritmici duri, produrre codice di quel livello può restare il collo di bottiglia. Il guadagno di tempo significa soltanto che ottenere in fretta una funzione dall'aspetto credibile non basta più a dimostrare che sia quella giusta. Quando l'esecuzione accelera, il fattore limitante si sposta, e con esso il valore: dove si concentra, ora?

Il limite si sposta sulla decisione: che cosa vale la pena costruire? Per rispondere bisogna leggere una richiesta insieme al processo di business che dovrebbe servire, ai dati che tratta, al codice e ai servizi con cui dovrà integrarsi, all'infrastruttura che dovrà sostenerla e all'operatività di chi ne gestirà gli errori. Bisogna riconoscere, in un requisito di una riga, l'ambiguità che costerà cara; sapere quale errore è recuperabile e quale no; rispondere delle conseguenze quando il sistema è in produzione alle tre di notte. L'accelerazione dell'esecuzione non risolve nessuno di questi problemi; permette soltanto di percorrere più in fretta anche una direzione sbagliata.

Questa è la visione d'insieme che il ticket affidato a Tommaso e Sandro non contiene. La richiesta delimita una modifica, ma il Senior deve vedere oltre: dove quella modifica atterra e che cosa cambia nel sistema. Non perché conosca in anticipo ogni dettaglio, ma perché attraversa i domini coinvolti con le domande che ne fanno emergere i confini.

L'agente può ricostruire dipendenze, esplorare il sistema e ampliare l'indagine, e in questo lavoro diventare indispensabile; può perfino proporre domande che il Senior non aveva considerato. Il contributo specifico del Senior sta nel riconoscere quali domande qualificano davvero il problema, quali risposte cambiano la direzione e quali conseguenze sono accettabili.

> **L'esperienza del Senior non è soltanto un archivio di risposte: è una gamma di domande con cui riconoscere il sistema nascosto dentro una richiesta locale.**[2]

[2] Leggere una richiesta locale come parte di un sistema più ampio, con i suoi processi, input, output e punti di rottura, invece che come compito isolato, è ciò che in inglese si chiama *systems thinking*.

Non è un aggiornamento, è una frattura

Lo spostamento del limite sulla decisione è reale; resta da capire di che natura sia. La generazione di codice guidata da un modello può sembrare un altro miglioramento degli strumenti a disposizione degli sviluppatori, una sorta di «IntelliSense 2.0». Non è così. Per chi, come me, ha legato il proprio contributo professionale all'esecuzione manuale, quella generazione produce una discontinuità molto più profonda: cambia il modo in cui si definisce il ruolo dello sviluppatore.

Linguaggi più espressivi, librerie e IDE visuali hanno già spostato molte volte il confine tra ciò che si scrive a mano e ciò che si affida agli strumenti. Un agente consente però di affidare a un sistema una porzione più ampia dell'esecuzione: esplora, propone e modifica artefatti attraverso più passi, oltre alla codifica autonoma.

La frattura di cui parlo riguarda il terreno sul quale hai costruito la tua identità professionale: la capacità di produrre codice. È proprio quel terreno a cedere. «Il codice lo scrivo bene e in fretta», qualità e velocità insieme, conta di meno come misura del tuo contributo. Saperlo fare resta necessario, ma non basta più a dire dove hai portato valore. Perdere il metro con cui ti misuravi è ciò che disorienta.

Si indebolisce l'idea del programmatore come **macchina che traduce il pensiero in codice**. Acquista più peso la responsabilità sull'intero percorso della soluzione: comprendere il problema, dirigere l'esecuzione, verificare il risultato e seguirne le conseguenze nel sistema.

Il mestiere non è scomparso: il suo baricentro si è soltanto spostato.

Perché le due tentazioni sbagliano bersaglio

Davanti a questo spostamento emergono due reazioni opposte. La prima è il **rifiuto**: continuare a scrivere tutto a mano per difendere il mestiere. La seconda è la **resa**: lasciare fare all'agente e limitarsi ad accettarne l'output. Sembrano incompatibili, ma condividono lo stesso errore. Continuano a misurare il valore sul terreno del codice.

Il rifiuto difende quel terreno proprio mentre, in molti casi, l'esecuzione manuale smette di essere il principale collo di bottiglia: vi si arrocca come se coincidesse con il mestiere. La resa presume invece che, tolta l'esecuzione, non resti altro da fare. Ignora una distinzione già posta nei Fondamenti: l'agente può esplorare alternative, formulare valutazioni e prendere decisioni operative circoscritte, ma la responsabilità finale e le decisioni direzionali designate non passano automaticamente insieme all'esecuzione.

Entrambe guardano ancora a quanto codice viene prodotto, proprio mentre la frattura chiede di valutare la direzione e le conseguenze.

Che cosa resta

Resta il giudizio.

È il principio da cui parte tutto il resto del libro: ***il valore di un Senior risiede nel suo giudizio, non nella quantità di codice che produce***. Sandro non è stato più bravo a scrivere codice: non ne ha scritto affatto. Ha riconosciuto che il requisito conteneva già una soluzione non ancora giustificata, e l'ha ricollocata nel sistema tecnico e di business dell'azienda. Il risultato non si misura in righe: il giudizio di Sandro ha orientato il lavoro.

Nella storia di Sandro i tre livelli si distinguono con nettezza. Ciò che deve restare vero per soddisfare il bisogno è il principio; interrogare la richiesta e confrontare le alternative è il metodo; la funzione CSV, la vista o l'integrazione sono implementazioni contingenti. Produrne una è soprattutto esecuzione, proprio il livello che oggi costa meno. Il giudizio, invece, attraversa tutti e tre i livelli e li tiene insieme: riconosce quale principio è in gioco, valuta i compromessi e assume la responsabilità della direzione scelta. Per questo la frattura non elimina il lavoro del Senior: lo porta in primo piano.

Il giudizio è meno facile da vedere del codice scritto: se ne osservano le decisioni, le ragioni e le conseguenze, non una quantità prodotta e misurabile. Passare un'ora a capire che stavi per costruire la cosa sbagliata, e non avere una riga di codice da mostrare a fine giornata, ti fa *sembrare* di non aver lavorato affatto.

Per anni hai imparato a sentirti produttivo quando l'editor si riempiva. La frattura ti chiede di disimparare quel riflesso e di trovare un altro modo, meno immediato, di riconoscere una giornata ben spesa: non da quanto hai prodotto, ma dalle scelte che hai orientato.

Una parola che dice troppo

«Giudizio» è però una risposta che apre subito un problema: la parola può voler dire troppe cose.

Una parte di questo giudizio sembra affiorare senza sforzo: una domanda posta prima ancora di sapere perché, un rischio intravisto ai margini del ticket, il ricordo impreciso di un sistema che si era rotto nello stesso modo. Sono segnali che vengono da anni di esposizione a sistemi reali, e che il Senior porta con sé anche quando non saprebbe formularli come una regola. Ma chiamarli «esperienza» non dice ancora perché a volte portano fuori strada.

Finché la parola tiene insieme fenomeni diversi, non si capisce che cosa distingua il giudizio da un'abitudine, né come il confronto con un agente possa renderlo più esplicito. Il capitolo che segue prova a separarli.

Il giudizio

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

L'esperienza fornisce i precedenti

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Il giudizio sceglie nel caso presente

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Perché il giudizio è situato

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Che cosa può portare l'agente

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Il confronto non è ancora un metodo

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

La cooperazione

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Che cosa manca a una conversazione per essere un metodo

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Quando la cooperazione si impoverisce

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Progettare il rapporto

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Il punto di contatto

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Il mandato

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Un compito nomina un'azione, un mandato rende esplicita la delega

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

La brevità non è il problema

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

L'unità della delega

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Mandato e specifica

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Dal perimetro allo spazio

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

L’autonomia

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

La domanda sbagliata

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Un’autonomia che si gradua

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Il perimetro definisce lo spazio

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Quando la decisione torna al Senior

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

L’arresto

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Fermarsi non è fallire

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Il limite va dichiarato prima

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Dal principio al comportamento

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Quando il mandato non basta più

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

La verifica

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

La plausibilità non basta

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Serve un oracolo con provenienza indipendente

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

La verifica si progetta prima

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Una prova non è ancora patrimonio

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

La persistenza

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Il giudizio, da solo, non sopravvive

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Conservare tutto non basta: la trappola dell'archivio

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Che cosa merita una traccia

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

I collegamenti contano quanto il contenuto

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Chi scrive la traccia

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

La persistenza non basta

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Che cosa diventa patrimonio

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Il contesto

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Ricordare non è il problema

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

La memoria conserva, il contesto seleziona

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Perché il contesto è la risorsa che conta

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Dal recupero all'amplificazione

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Parte 2 — Costruire i binari

La Parte 1 ha tenuto le tecnologie di riferimento sullo sfondo, perché descriveva il modello operativo con cui Senior e agente lavorano insieme, non gli strumenti su cui poggia. Chi vuole intervenire sul proprio ambiente agentico ha però bisogno di sapere su che cosa mettere le mani; per questa ragione da qui in avanti il libro chiama le cose con il loro nome.

I capitoli che seguono tracciano le coordinate: prima che cosa si governa, poi chi lo fa e con quale forza, infine la verifica sul lavoro svolto.

- **Il cappello** apre la seconda parte del libro: che cos'è un harness, dove finisce il framework e comincia il System Harness, e le quattro domande che guidano ogni capitolo.
- **La cerimonia** dà forma all'unità di lavoro: apertura dichiarata, chiusura validata, scrittura dosata in corsie distinte.
- **La fonte** governa ciò che entra dall'esterno: ne registra la provenienza e la distilla in conoscenza o decisione.
- **Il grafo** fissa la struttura che la cognizione assume una volta messa per iscritto, con i suoi nodi, i suoi archi e i suoi confini.
- **Il prompt**, **le skill**, **gli strumenti**, **la sessione** e **la porta** percorrono i punti di estensione, dal testo che prescrive all'hook di Git che impedisce l'azione.
- **La traccia** compie il movimento di ritorno: ciò che è stato scritto torna recuperabile per significato.
- **Il laboratorio** è il caso di studio della parte: una sessione sola, dall'inizio alla fine, in cui le superfici lavorano insieme su un progetto reale e circoscritto.
- **La prova** collauda l'harness per quello che è, software: i test verificano il metodo, non il prodotto.
- **Il bilancio** dichiara che cosa l'esperimento permette di affermare, con i suoi limiti e le sue evoluzioni.

Un'avvertenza prima di cominciare. Questo testo non è il manuale di SAF, e le pagine che seguono non chiedono di scaricare nulla: non c'è un sorgente da clonare, e nessun

capitolo documenta un'installazione. Un repository, però, esiste ed è pubblico: saf-linkcheck, il laboratorio in cui Senior e agente hanno messo alla prova l'harness a partire da una cognizione vuota. Lì l'harness è installato, e con lui i file che questi capitoli citano: il prompt del metodo, le skill, gli hook di Git e la configurazione della sessione. Ogni file che vive nel laboratorio compare nei riquadri d'estratto con il proprio nome come collegamento: in pagina restano le poche righe che contano, il file intero e ciò che lo circonda stanno nel repository. Lì si trova anche il caso per esteso — i record, i commit rifiutati, la cronologia della sessione — di cui il capitolo *Il laboratorio* fa il racconto. È materiale di prova, non un modello da copiare; chi non apre il repository non perde nessun passaggio del ragionamento.

Glossario

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Accesso semantico

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Agente

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Autonomia

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Binario di scrittura

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Cognizione persistente

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Contesto

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Context rot

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Distillazione

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Framework agentico

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Giudizio ingegneristico

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Giudizio situato

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Implementazione

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Mandato

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Memoria

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Metodo

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Modello linguistico

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Modello operativo

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Oracolo

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Pair

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Principio

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Punto di estensione

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

SAF (Senior Amplification Framework)

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Scheda

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Senior amplificato

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Sistema agentico

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Skill (nel metodo)

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

System Harness

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.

Traccia cognitiva

Questo contenuto non è disponibile nel libro di esempio. Il libro può essere acquistato su Leanpub a https://leanpub.com/il-senior-amplificato.