曖昧さから確実性へ

この情報は少し頭を悩ませる内容です。読む前に深呼吸することをお勧めします。

序文

仕事中、私自身も周りの人も様々な苦境に陥ることがよくあります。具体的な現象は多岐にわたります。

私はこのことについてよく考えた結果、これらの問題と困難は「曖昧さ」と「確実性」という 2 つの概念に起因することが分かりました。

それをクリックしてください。

IT 実践者として、私たちは需要分析から設計、プログラミング、デバッグ、配信までのサイクルを常に実行しています。

各段階は、漠然としたものから明確なものへと進むプロセスです。私たちは仕事の対象を明確に理解しており、心理状態も明確です。

あなたは安らぎを感じるでしょう。これは確信を得る心理状態です。

プロジェクト開始時の混乱からシステム納品時の秩序に至るまで、漠然としたものから確かなものまでと表現するのが適切でしょう。

作業を小さな単位に分割した場合も、同様の変化が起こります。

1ヶ月の仕事も同じです。コード1行、Javaクラス1つ、関数1つ、システム1つ、すべてが

学習の過程でも同様の変化のプロセスが経験されてきました。

しかし、これを達成するのは容易なことではなく、その過程では不安や挫折、悟り、安堵が生まれるでしょう。

どうすれば不安やフラストレーションを軽減し、できるだけ早く悟りと安らぎを得ることができるでしょうか。これが私がこの資料を書いた目的です。

これはプログラミングだけでなく、世の中のあらゆることに当てはまります。宇宙の進化、国家の変化、そして人々の関係性の進化。

確かにそうです。プログラミングは特別なケースに過ぎません。曖昧さと確実性の弁証法的な関係をどう理解するか。

曖昧さから明確さと確実性への橋渡しをし、効果的な思考モデルを確立するにはどうすれば良いかを考えてみましょう。

『道徳経』の有名な引用：「語られる道は永遠の道ではない。名付けられる名前は永遠の名前ではない。名付けられないものは万物の始まりであり、名付けられたものは万物の始まりである。」

「万物の母」。実は、それが私たちの進むべき道を示してくれたのです。そうでしょう？ 無名とは曖昧な状態です。

命名とは、物事に名前を付けることです。プログラミングでは、ファイル名、変数名、クラス名など、多くの名前を付ける必要があります。

名前、メソッド名、システム名、関数名、モジュール名。プログラミングのアイデアには、デザインパターン名、書籍名、アルゴリズムが含まれます。

名前、データ構造名、データベーステーブル名、インターネットプロトコル名、TikTokアカウント

名前。

命名とは思考プロセスの段落の要約です。そして名前は、思考プロセスを表すシンボルです。

概念と、一連の認知的結果。名前もまた、私たちのコミュニケーションに欠かせないものです。

「体の要求を満たすために食べ物を口に入れる」という長いリストの代わりに、「米」という2つの単語が使われました。

なぜなら、これら 2 つの単語の意味は確かであり、聞き手もこれについて合意していると確信しているからです。

たとえば、名前がなければ人々はどのようにコミュニケーションをとるのでしょうか。

したがって、名前を付ける能力を習得していれば、実際には思考を漠然としたものから明確なものへと変える方法を習得していることになります。

ファジーという名前自体は、あなたが「私はあいまいな状態にあるだろうか？」と考えるきっかけとなるものです。

曖昧さから確実性までの各ステップに名前を付け、次の章で説明していきます。

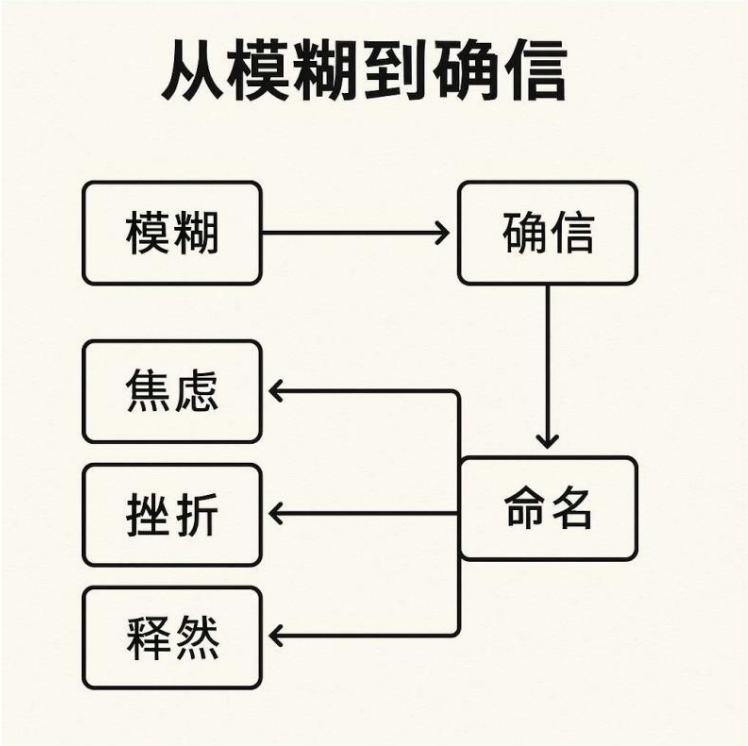
しかし、この情報だけでは不十分であり、より詳細な情報が必要です。

これらの内容は、一連の資料全体を通してのみ確認することができます。後続の資料を読んでも

世界は広い。だからこそ、読者の皆さんがこの記事の情報を活用していただければと思う。

この文書で確立された基本的な方法は、自分の分野における独自の信念を探求することであり、それがこの文書の意義でもある。

どこ。



曖昧さは当たり前だが、確信する必要がある

私たちは不確実な世界に生きています。

プログラマー、デザイナー、プロダクトマネージャー、学生、起業家の皆さん、日常生活が未知のことや曖昧なことでいっぱいではないでしょうか？

変化するプロジェクト目標、複雑なコードロジック、不完全なドキュメント、あいまいな要件、予期しないバグ、あいまいな上司。

未知数、時間は決して十分ではありません...このような環境では、「すべてが明らかであり、すべてが確実である」というのは単なる幻想です。

曖昧さが当たり前です。

しかし、誰もがその曖昧さの中で停滞するわけではないことも分かりました。

深呼吸をしてすぐに状態に入り、情報を確認したり、質問したり、書いたりすることで曖昧さに直面する人もいます。

彼らは、混沌とした情報を

情報を実行可能なステップに変換し、曖昧な問題を明確な判断に変えます。

この能力は、曖昧さから確実性へと移行する能力です。

曖昧であることは悪いことではないが、長い間曖昧なままでいると停滞してしまう

曖昧さ自体は恐れるべきものではありません。それは探求への入り口であり、思考の出発点であり、認知の亀裂を抜ける成長への道なのです。

しかし恐ろしいのは、私たちの多くが曖昧さを終わりだと勘違いしていることです。

「準備ができたらやります。」

「ちょうどいい感じです。」

「いずれにせよ、誰も確信が持てない。」

「問題ないはずだよ。」

これらの言葉は普通に聞こえるかもしれないが、その背後には曖昧な状態の放棄がある。

ぐるぐる回りながら、進歩について考えず、あえて質問せず、挑戦しようとせず、彼は自分の認識を浮遊の低いレベルにしっかりと結びつけた。

状態。

時間が経つにつれて、彼は物事をするのがだんだん遅くなり、自信がなくなっていき、さらには自分自身を疑い始めることに気づくでしょう。

能力。

確信は認知の「アンカー」である

曖昧さが霧のかかった森だとしたら、確実性は私たちがそこに設置する錨です。

ポイント。

明確に理解できるすべての点、説明できるすべての概念、そして決定を下すことができるすべての判断は

認知世界における「アンカー」。

これらのアンカーがあれば、混沌の中で徐々に構造を構築し、未知のものの中で方向感覚を養い、複雑な状況を乗り越えることができます。

制御性を確立するために、各アンカーに名前を付けます。

スレッドの概念により、エラーがスレッド同期の問題によって発生したことが明確にわかるでしょう。

ビジネスプロセス図を描いて新入社員に説明することができます。

新しい API の呼び出しチェーンを理解しているため、新しい API が既存のロジックを壊さないことがわかります。

この「知っている」「確認した」「明確に説明できる」というのが確信感です。

「私は事実を知っている」ではなく、「私は自分が知っていることを知っている」のです。つまり、自信、方向性、そして行動力があるということです。

力。

第 1 章 曖昧性と確実性とは何か?

私たちはよく「ちょっと混乱している」とか「きっとこれが正しい」と言います。

しかし、曖昧さとは一体何なのでしょうか ?そして確実性とは何なのでしょうか ?

これら 2 つの単語は反意語のペアのように聞こえますが、実際には、認知の連続体のようなものです。

脳の世界に対する理解は、このスペクトルに沿って常に変化しています。

---

1. 「ファジー」とは何ですか?

曖昧さは「分からない」という意味ではなく「言えない」という意味です。

頭の中に概要や印象はあるようですが、それをはっきりと表現したり、判断したり、決定したりしたい場合は、

彼はためらい、迷い、曖昧になり始めた。

簡単な定義:

曖昧さとは、物事に対する明確な理解、判断、位置づけが欠如している状態です。

あいまいさの典型的な特徴は次のとおりです。

特徴	説明する	例
そうかどうかは分からない		この機能には副作用はありますか? 使用 ?"
曖昧さ	このようにも、あのようにも理解できる 理解する	「この『速い』というのは1秒ですか、それとも1時間ですか ?」 時間 ?"

特徴	説明する	例
	基準や根拠がなければ判断できない	「このデータは正常ですか？」
	論理的な結論の欠如 情報はそこにあるが、理解できない。	「なぜこのプロセスはぐるぐる回ってまた戻ってくるのでしょうか?
構造	系	「

プログラマーの例:

Javaプロジェクトを読んでいて、UserManagerクラスがメソッドを呼び出すのを見ます

ハンドル（）。

分からないこと:

- •このメソッドはログイン、登録、ログアウトを処理しますか?
- •同期ですか、それとも非同期ですか?
- •依存関係は何ですか?

handle() が何であるかを知らないわけではなく、単に漠然としているだけです。

---

2. 「確実性」とは何でしょうか?

確信するということは、「覚えている」ということではなく、「理解し、明確な判断ができる」ということです。

それが何なのか、なぜ生まれたのか、何ができるのか、そして他の人に伝えることができるようになったら、

あなたは「確信」の状態にあります。

簡単な定義:

確信とは、明確で根拠があり、構造化された理解の状態であり、判断力、責任感、さらには

他の人を指導する。

自信のいくつかの特徴:

特徴	説明する	例
明確な判断	"それが現実さ"	「このクラスはセッション状態の管理を担当します」
壊す		
説明可能であり、全体のストーリーを説明できる		「モジュールXを呼び出して権限情報を取得します。
		息”
繰り返して他の人に明確に説明することができます。		「このプロセスを示すために図を描いてみましょう」
拡張可能	同様の問題を解決するために使用できる	「このパターンはログ管理モジュールで使用できます」
	質問	

引き続きプログラマーの例を使用します:

handle() の実装を詳細に分析し、次のことを知りました。

- •ユーザー登録を処理します。
- •データベースを呼び出して電子メールを送信できます。
- •構成パラメータとスレッド プールに依存します。
- •エラー処理戦略は何ですか?
- •再利用する場所。

これで、ドキュメントの作成、ロジックの変更、そしてモジュールの動作を新規開発者に説明できるようになりました。自信がついたと言えるでしょう。

III. 曖昧さと確実性の関係 :認知の道

それらは白黒はっきりしたものではなく、連続的に進行するものです。

曖昧（わからない、不確か)→半分理解→理解→明確→確信

初めて触れるときには曖昧な部分もあるので、ゆっくり理解していきましょう。

•文書を読む（補足情報）

•図面（建築構造）

質問する（訂正）

実験（検証）

•ナレーション（逆確認）

徐々に確信へと変わっていきました。

要約: 曖昧さと確実性の違い

次元のぼかし 信念

根拠や判断が不確かで不確かで自信がないと感じる

はっきりと表現できない 明確に説明できる

ためらいがちな行動、試行錯誤、先延ばし、決断力、明確な計画

他の人を混乱させ、他の人を導くことができる

曖昧さが続くと忘れてしまうことが多く、確信が内面化されてしまうことがよくあります。

章の要約

•曖昧さはあなたのせいではありません。それは不完全な認知の正常な状態です。

自信は才能ではなく、練習を通じて獲得できるものです。

•曖昧さは霧であり、確実性は地図である。

•この本は、霧を晴らし、明確な地図を描くのに役立つように設計されています。

第2章 :曖昧さから確実性への変化

曖昧さから確実性への変化は認知能力の向上のプロセスであり、脳が明確なモデルを徐々に構築していくプロセスでもあります。

これは、訓練して繰り返すことができるプロセスとして理解できます。

程。

1. 概要 :曖昧さから確実性への道

混乱（混乱、ためらい、圧倒される気持ち）

↓

ぼんやりと見える（何かが理解できないことに気づく）

↓

良い質問をする（問題の核心を定義する）

↓

情報の探求（読む、質問する、実験する）

↓

モデルを構築する（整理、要約、一般化）

↓

モデルを検証する（テスト、適用、反映）

↓

保証（明確さ、安定性、自信）

このプロセスを「6段階認知明確化法」と呼ぶことができます

2. 6つの段階の詳細な説明

ステップ1 :ぼかしを確認する

「何かが理解できないようなのですが、それが何なのかがわかりません。」

方法：

質問を書き留めてください（「Xが何なのかわかりません」）

あなたの不安を説明してください（「私が間違ってしまうのではないかと恐れているのは…」）

目的: 「曖昧な存在」に気づくことが第一歩です。

ステップ2: 良い質問をする

私は何を理解しようとしているのでしょうか?

方法：

漠然とした質問を具体的なものにしましょう: 誰が? 何を? なぜ?

例えば、5W1H法（何を、なぜ、どのように…）を使って問題を分解する

「もしも」で境界を探る

例: 「Javaが書けない」ではなく、「メインメソッドがプログラムを開始する方法が分からない」の"

ステップ3: 情報を見つける

他の人はこれをどう理解しているのでしょうか？彼らから何を学べるでしょうか？

方法：

ドキュメントを読んだり、コードを見たり、人に質問したり、ChatGPTを確認したり

完璧な答えを求めるのではなく、まずは「予備的な理解」を築く

注意：入力しすぎると曖昧になってしまうので、学習しながら要約する必要があります。

ステップ4: モデルを構築する

「私の理解を表明してもよろしいでしょうか？」

方法：

自分の言葉で話す（ファインマンテクニック）

絵を描く／メモを書く／例を挙げる／類推を書く

断片化された情報を構造化（チャート、フローチャート)に変換する

ヒント: 「なぜこれがで、あれではないのか」という問いを使って、自分自身に考えさせるようにしましょう。

ステップ5: モデルの検証

「私の理解は応用できますか？」

方法：

コードを書き、実験し、他の人に検証してもらう

理解度をテストするために、意図的に極端な状況を作り出す

多角的なドリル（逆推論、境界値テスト）

例: メイン メソッドのさまざまなバリエーションを記述し、どれがコンパイルされ、どれがコンパイルされないかをテストします。

ステップ6: 確認

「なぜこのようなことが起こるのか、何が問題なのかがわかっています。」

パフォーマンス：

自分の判断の根拠を明確に表現できる

困難に直面しても簡単に動揺してはいけない

他人に教えたり間違いを認めたりできる

3. それは「スキル」であり、瞬間的なひらめきではない

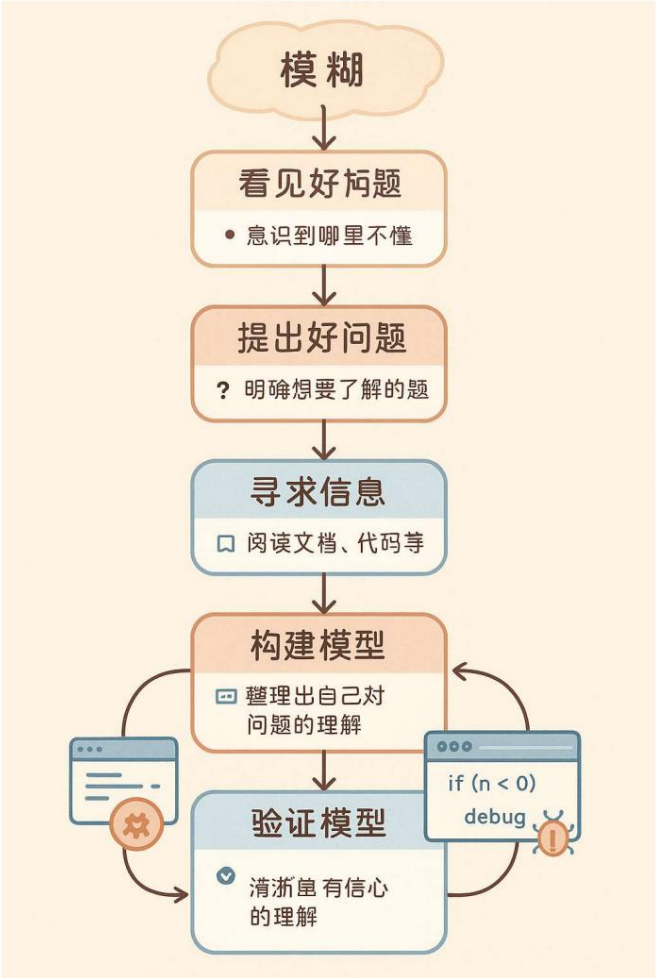
曖昧さから確実性へ、それは天国への飛躍ではありません。

代わりに、「曖昧さの特定 → 問題の定義 → 理解の構築 → モデルの検証」という繰り返しのトレーニングを通じて行われます。

練習する。

それはまるで、筋肉トレーニング、認知的ウェイトリフティングを何度も繰り返すようなものです。

最終的には、混沌の中でも明瞭さを維持し、複雑さの中でも構造を見ることができるようになります。



第3章 ぼんやりと見る

「確信を得るための最も難しいステップは、まだ曖昧な状態にあることを認めることです。」

これは、多くのプログラマー、エンジニア、マネージャー、さらには一般の学習者でさえも長い間認知の泥沼にはまってきた点に当てはまります。

中核的な問題

やり方がわからないとか、リソースがないとかいうのではなく、私の心の中では「ずっと前に理解しておくべきだった」という声が聞こえてくるのです。

これには実際には 2 つの絡み合った罠が存在します。

1. 偽りの確信

「わかっていると思うよ。」

「以前見たことがある」

「これの何が難しいの？」

「そろそろ頃合いかな」

これらは表面的には「自信」があるように見えるかもしれませんが、その背後には実際には検証されていない漠然とした認知的仮定があります。

他の人から詳細を尋ねられたり、動機を質問されたり、フローチャートを描くように求められたりすると、あなたの正体は暴露されてしまいます。

2. 感情的な抵抗

「理解していないことを認めたくないんです。」

「今聞いたら、プロ意識がないと思われるだろうか？」

「私は10年間コーディングをしてきました。もしやり方が分からなかったら恥ずかしいです。」

この抵抗は、「私は今、実はぼんやりしている」という重要な言葉を発することを妨げます。

そこで、「私は知っているはずだ」というイメージを維持するために、私たちは次のことを行います。

文書をチェックしないでください

他人にアドバイスを求めない

理解したふりをする

書き留めてください

結局、崩壊するか、やり直さなければならなくなります。

「曖昧さを認める」というのはなぜ難しいのでしょうか？

レベルにおける抵抗の源

認知レベルでは、実際にどれだけ理解しているかは不明瞭である

感情的な顔、恥、自己イメージの不安

ソーシャルメディアのユーザーは、自分の欠点をさらして信頼を失うことを心配している

専門家は「無能」や「非専門的」だと思われることを恐れている

3.1 曖昧なケース

鍵を見つける事件

鍵を探すというケースがあります。娘が鍵をなくしたと言っていたので、「よく探してね」と言いました。

どこも探し回ったけど見つからなかったと彼女は言いました。「もしお父さんが見つけたらどうするの？」と聞いたら、彼女は信じられないと言いました。

見つけられますよ。結果的に、彼女のバックパックの中に鍵を見つけました。彼女は、私がここに置いたことは一度もないと言っていました。

真ん中だよ。腹立たしいと思いませんか？

ハハハ、確かにそれは真実であり、面白くて悲しい、そしてそれは「誤った確信」の完璧な例です。

+ 曖昧な盲点 + 認知的閉ループ"のケース、

詳しく見てみましょう:

ケースレビュー :鍵を見つける認知ドラマ

娘は鍵を探したと言って、念入りに探したつもりだったと言いました。しかし、それは間違いでした。娘は鍵を探したと思っていたのですが、実際には探していませんでした。

真の確認

彼女の父親は、探すときには心をオープンにしておくようにと彼女に思い出させました...

娘は反論する。「あなたも見つからないわ。」認知的閉鎖（結果の特定）認知的閉鎖：「何もない」と誤って信じる

見つかった = 検索した = 見つからない"

父親はリュックサックの中に鍵を見つけ、実際に行動で仮説を検証した

娘は「ここに置いたのは私じゃない」と弁明した。

制御機構

この症例で明らかになった認知の問題

1. 誤った保証:

彼女は徹底的に調べたつもりで、「印象を受けた」ことを「検証した」と勘違いしていた。

キーワード: 「確かに見た」「ここにあったとは覚えていない」「早朝に探した」

→ 実は検証もせず、記憶や印象だけを頼りにしていました。

2. 認知上の盲点：

彼女は「一時的に習慣を変えて間違った場所に置くかもしれない」という可能性をまったく考えなかった。

自分自身の行動についても混乱が生じます。

3. 自己完結的な認知ループ：

彼女はまず「鍵はそこにない」という結論に達し、次に「探した」を使ってこの結論を反証し、閉じたループを形成しました。

指輪。

見つからなかった→だからそこにはない→だから探さないで→見つからない→私が正しい

→ 論理的自閉症

4. 認知防衛機構：

それを見つけると、彼女は「ここに置いたのは初めて」と言います。これは認知的（不協和的）処理メカニズムです。

「私は間違っていない」というイメージを維持してください。

これがプログラマーであれば、結果は次のようになるでしょう。

「コードを確認しましたが、明らかに問題はここにはありません」

→ 結果的に、ログを追加して試してみると、問題はここにあることがわかります

→ 彼は答えました。「これまでここで問題が発生したことはありません…」

馴染みがある感じですか？

ケース2

「何かを感じる」と言う人もいますが、実際には彼の感情がどこから来ているのかわかりません。もしかしたら、

寝る。

例えば、SWIFTシステムに携わったことがある人なら誰でも、Swiftメッセージには次のような類似点があることを知っているでしょう。

MT300、MT202、どちらもTag56を持っています。

その結果、彼はMT202で実行されるべき操作を、MT300で実行した。

見せる。

このぼやけは何なのでしょう？

これは非常に典型的で極めて危険な「類推的曖昧性」です。

曖昧さ（曖昧さ）は、検証されていない直感的な操作と過剰な認知の転移を伴う。

これは、「曖昧さから確実性へ」の道筋における、高度な曖昧さの誤解の範疇に入ります。

このケースの核となる動作:

MT300とMT202はどちらもTag56を持っているので、同じように処理できると思います。

その結果、MT202 の動作ロジックが検証なしに MT300 に「適用」されました。

また、リクエストや二次確認もありませんでした。

曖昧型判断 :類推的曖昧さ＋直感的確実性＋検証の欠如

ファジィ型表現 危害

類推モデル 「類似構造」を「同一線」と捉える システムが誤って操作されると、全く異なる動作をする可能性があります

ベースト のために" 深刻な

ファジィ型表現		危害
直感的	「できるはずだと思う」けど説明できない	再現不可能、検証不可能、無責任
手紙	なぜ	任命する
検証不足	ドキュメントなし、テストなし、リクエストなし	アクションの前にエラーがインターセプトされない
	見せる	

アナロジカルファジィの構造的特徴

類推が有効となるための条件（認知科学の観点から）誤った類推におけるバイアス

共通の構造を持つ + 共通の意味を持つ + 持つ	コンテキストを無視して、表面フィールドの類似性（タグ56）のみを参照する
類似の用途	およびビジネス目的
移行ロジックは検証可能	検証を省略し、「経験」や「感覚」に頼って判断する
明確な違いの境界線がある	MT202 がクリア指示であり、MT300 がデリバティブ指示であることに気づきませんでした。
	商品取引確認

ケース3

経験の再利用という問題もあります。部下の一人が、別の現場での経験を活かしていました。

その結果、彼は私の許可なく私の職場でも同じことをしました。

悪影響をもたらす操作。どうすればこれを防ぐことができますか？

このケースは非常に一般的であり、職場での「無意識の経験の再利用」のリスクを明らかにしています。

この種のエラーは「経験」の兆候のように見えるかもしれませんが、実際には、文脈の検証なしに経験を盲目的に転送した結果です。

認知バイアス。

問題の本質 :経験の伝達が制御不能になっている

次のように分類できます:

ファジー型 :文脈検証なしの経験の伝達

ケース分析

シーンの動作		の結果として
アイテムA 目	部下が複数のファイルをアップロードしましたが、システムが正常に処理し、すべてがうまくいきました（経験の蓄積）	
オプションB 目	彼は同じものを再利用した 操作する	不正なプロセス、データ汚染、または権限がトリガーされた 一線を越える

なぜこれが「経験済み」ではなく「漠然とした」ものなのでしょうか?

質問の種類	説明する
型	
コンテキストモデル ベースト	彼は、異なるプロジェクト環境におけるシステム設定、プロセス、ルールが一貫しているかどうかを理解しませんでした。
境界モジュール ベースト	彼は何が「許容される」ことであり、何に許可が必要なのかを知りませんでした。
検証がありません	彼は「複数のファイルのアップロードが許可されているかどうか」、ホワイトリストやパスルールがあるかどうかについては確認しなかった。
失う	バージョン要件など

彼は「経験主義者」ではなく、経験の一般化者であり、経験は新たな文脈で検証されなければならない。

安全に移行できます。

なぜこれが危険なほど曖昧な点なのでしょうか?

- 「積極的」に見えるが、実際には「無許可で権限を逸脱している」

彼は自分が正しいことをしていると考えており、後から「以前もそうしていた」と言って自分を弁護するかもしれません。

これは行動上の問題ではなく、認知モデルが更新されていない問題である。

このような経験の伝達における曖昧さを防ぐにはどうすればよいでしょうか?

1. 組織レベルで「文脈的再検証メカニズム」を確立する

物体	予防戦略
経験豊富なスタッフ 仕事	教育する :経験は直接伝達することはできず、シナリオやシステムによって確認されなければならない
全従業員	プロジェクト間の操作について明確な「許可境界」を確立する。「このような操作は承認されなければならない。 許可する"

プロジェクト文書には、どの行動が「経験の移転を許可」され、どの行動が再評価される必要があるかが明確に記載されています。

---

2. 経験移転のための「5つの確認」を策定する

5つの質問による経験移転検証方法：

1. 元のシナリオと現在のシナリオの違いは何ですか？（システム、バージョン、権限、目標）
2. 元の操作にはコンテキスト依存関係がありますか? (前提条件は一貫していますか?)
3. 現在のシーンには明示的な許可がありますか?
4. 小規模テストを実施しましたか?
5. プロジェクト リーダー/チームに通知され、提出されましたか?

経験の転送は、5 つの質問すべてが満たされた場合にのみ安全です。

ケース4

同僚は「前回も同じことをしたのを覚えています」と言いました。

デプロイメント ログを確認したところ、環境はベータ版、バージョンは v1.1 で、構成にはデフォルトで有効になっているフラグがありました。

使用;

今回は本番環境のv2.0で、フラグはデフォルトでオフになっているので当然違います。

彼はふざけていたわけではなく、ただ記憶の錯覚にとらわれていただけなのです。

彼はある断片を覚えていたが、鍵となる条件を忘れていた。

彼は結果は覚えているが、状況は忘れている。

彼は「思い出した」と思っていたが、実際には彼の心の中には「不完全な地図」しか残っていなかった。

この事例は、非常に微妙ではあるものの、非常に一般的な認知的曖昧性のもう一つのタイプを正確に指摘している。今回は、

それはこう呼ばれます:

記憶の錯覚の曖昧さ

別名：「断片的記憶の欠陥のある再生」

例の構造は次のとおりです。

同僚は「前回もこのようにやったのを覚えているよ。これで正しいはずだよ」と言いました。

•実際には、彼はいくつかの手順や状況しか覚えておらず、今回の状況は異なっていました。

•結果: 同じ操作を実行しましたが、異なる結果が得られ、期待は満たされませんでした。

この種の曖昧さの核心的な問題は「彼は知らない」ということではなく、

しかし、彼はそれをはっきりと覚えていると思っていたが、実際には重要な部分を忘れていたか、あるいは2つの状況を混同していた。

。

この曖昧さには主に 3 つの要素があります。

ファジー次元表現

メモリの断片化
「キーアクション」だけを覚えておき、隠れた依存関係（環境変数、設定など）は忘れてください。
アイテム）

記憶の混乱
2 つの異なるイベントを混ぜる - 「前回はシステム A でしたが、今回はシステム B です」
「システム」ではなく、心は統合されている

記憶は間違いにつながる

自信
「以前成功したことがある」という理由で、現在の業務に誤った自信を抱いている

一般的な症状:

「このようにデプロイしたのを覚えている」→デバッグモードだったことを忘れていた

「前回も同じことをしました。」→データベースに権限制御がないことを忘れていました。

- 「インターフェースは以前このように呼び出されました。」→トークンの有効期限が切れていないことを忘れていた
- 「パスは/data/であるべきだと覚えています。」→ソフトリンクを追加したことを忘れていました

本質的には、これは「想起主導の確認バイアス」です。

原因と結果

記憶が現実のように感じられる → 真実だと信じる → 検証ステップをスキップする

成功体験を間違って覚える → 新しいタスクを真似する → 気づかないうちに失敗する

環境は変わったのに、運用は同じままなので「なぜ今回はうまくいかなかったのか？」という疑問が生じます。

「記憶の錯覚」によるぼやけを防ぐには？

方法	説明する
「覚えている」ではなく実際の操作を記録する → 「操作ログ/READMEを確認した」	
バージョン番号、パラメータ、パス、構成ステータスなど、各実験の実験条件を書き留めます。	
検証プロセスを開発してください。たとえ「覚えている」としても、現在の環境に適用可能かどうかを確認する必要があります。	
自問自答しながら決断するのは避けましょう。「覚えています」と言う代わりに、「今回は何を確認しましたか？」と自問しましょう。	
スクリプト、ドキュメント、データ サンプルなど、すべてをバージョン管理します。	

記憶は幻想であり、

自分自身を騙すのが一番簡単です。

明確さは思い出すことからではなく、再構築と検証から生まれます。

上記の症例に共通するのは、いずれも自分が曖昧な状態にあることに気づいていなかったこと、つまりメタ認知の欠如です。

失う。

毛沢東はかつて、人間にとって最も大切なのは自己認識だと述べました。自己認識とは、自分自身を正しく評価する能力のことです。

力。

これは空論ではなく、「認知的自己調整能力」の極めて高い価値を明らかにしています。

本当に賢い人とは、「常に正しい」人ではなく、「自分がどこで間違っているかを知っている」人です。

経験に囚われないでください。感情に囚われないでください。アイデンティティに縛られないでください。

自己認識の最初の行為は、「今は理解していないかもしれない」と言う勇気です。

この発言の心理的コストは非常に高い。

「プロフェッショナル」、「経験豊富」、「信頼できる」という仮面を脱ぎ捨てる必要があります。

「再出発者」の姿勢で仕事、判断、システム、人に立ち向かおう。

しかし、このステップこそが、曖昧さから確実性への唯一の入り口なのです。

古代中国には、剣を探すために船を彫る、鈴を盗むために耳を塞ぐ、木の切り株でウサギを待つなど、多くの寓話があります。これらはどれも、混乱し、何をすべきか分からない状態を描いています。

これは知識の皮肉です。私たちは常に冷静さを保ち、この危険な認知の罠に陥ってはなりません。

誰もが間違いを犯す可能性を常に意識していれば、成長はそう遠くないと信じています。

残っているのは、以下に説明する方法を正しく使用することだけです。

第4章 質問の仕方 - 曖昧さを明確な言葉に圧縮する

なぜ質問することから始めるのでしょうか?

なぜなら：

質問が漠然としていればいるほど、より一般的な情報が見つかります。

質問が明確であればあるほど、他の人があなたを助けやすくなり、あなた自身も答えを見つけやすくなります。

質問が具体的であればあるほど、認知モデルは成熟します。

一言で言えば:

「質問の仕方を知らない人は、自分が何を理解していないのかを知りません。」

プログラマのための4段階質問法 :SPQRモデル

ステップ	名前	意味	例
S	状況 地域）	現在の問題は どこですか？	「Spring Boot プロジェクトで構成をホットアップデートします」
P	問題（現在 象）	具体的な出会いとは 異常な？	「application.yml を変更しましたが、設定が反映されませんでした」

ステップ	名前	意味	例
ステップ 質問	質問（聞く）	何を知りたいですか？何？	「なぜ設定が自動的に再読み込みされないのでしょうか？何かきっかけがあるのでしょうか？」機構？"
R	研究（試す）	確認/試しましたか何？	「アクチュエータ/リフレッシュを試し、構成を試しましたリロードしても効果なし

—

質問する前にこれらの 4 つのポイントを書き留めてください。質問は次のようになります。

「なぜ設定変更が無駄になるのか？」

なる：

「Spring Bootプロジェクトでxxxを設定するためにapplication.ymlを使用しました。変更後、

設定が反映されませんでした。アクチュエータを更新しようとしましたが、更新がトリガーされませんでした。どうすればよいでしょうか？

リロードをトリガーする他の方法はありますか？」

—

これは、「あいまいな言語」から「構造化された思考」への移行の始まりです。

典型的な「悪い質問」と「良い質問」の比較

悪い質問、良い質問

「プログラムエラー何？管理？"	メソッドAを実行すると、NullPointerExceptionがスローされます。コールスタックは次のとおりです。どの変数が null であるかは不明です。"
「この機能は使えますよ。」	「addUser() を呼び出すと false が返され、ログを確認するとデータベース接続が失われたことがわかります。エラーが失敗した場合は、構成項目 DB_URL が正しいかどうかを確認してください。

悪い質問 良い質問

「私は 「ローカルにデプロイしたSpring Bootインターフェースの GET /userは404を返しますが、

「アクセスできません」 コントローラーに@RequestMappingが追加されました。パスでは大文字と小文字は区別されますか？

—

何か「不明瞭」なものを見つけるたびに、それはまだ曖昧な状態にあることを意味します。

—

質問支援ツール :曖昧点位置表

私の現在の混乱は… セルフチェック質問の例

入力はどのように行われますか? パラメータのフォーマットは正しいですか ?ヘッダーは正しく設定されていますか？

システムの現在の状態はどうですか ?キャッシュはありますか ?最新のコードがデプロイされていますか？

何を期待していたのでしょうか ?実際はどうだったのでしょうか ?私の期待は間違っていたのでしょうか ?それとも、行動は変化したのでしょうか？

何を試しましたか ?結果はどうでしたか ?ログメッセージはありましたか ?問題を再現できますか？

—

読者は「質問の下書きを書く」習慣を身につけ、「頭の中にある漠然とした質問」を「紙に書いた質問」として書き留めることが推奨されます。

構造"。

段落の提案

- 「明確な質問は明確な認識の最初の試金石である」
- 「言語は曖昧さを圧縮するためのツールである」
- 「まず、何が分からないのか教えてください。」

質問の質がどこまでできるかを決める

質問することで混乱を整理することができます。

良い質問をすることができれば、答えに近づいていることになります。

質問が具体的であればあるほど、理解がより明確になります。

「一つの質問をするのではなく、一連の質問をすることが大切です」

質問すること自体は、漠然としたものから明確なものへと向かうプロセスである

「最初の質問」はなぜ曖昧になることが多いのでしょうか?

1. あなたの認知はまだ混乱しているため、既存の語彙を使ってそれを表現することしかできません。

2. 「現象、原因、期待、境界」をまだ区別していないため。

3. 実際に何を聞きたいのか分からないからです。

—

例えば：

最初の質問は、「なぜサービスが開始されないのか？」です。

合理的に思えますが、実際には次のような曖昧な点があります。

どんなサービスですか?

起動中に何が起こりますか?

ログは何を示していますか?

•前処理は行いましたか?

•どのようにパフォーマンスを期待しますか?

—

この時点ですべきことは、「不安に駆られて答えを待つ」ことではなく、自分自身に問いかけることです。

•私の質問は具体的ですか?

•完全に説明できましたか?

•現象について尋ねているのか、それとも原因について尋ねているのか？

•私が間違った質問をしている可能性はありますか?

5段階の漸進的質問モデル（漠然としたものから確かなものまで）

待って クラス	質問 型	特徴	例
1	曖昧な質問 質問	範囲が広く、漠然としている 詳細	"なぜだめですか？"
2	驚異的 質問	特定の行動を指摘したり、 症状	「起動時にエラー メッセージが表示され、ログにはポートが占有されていることが示されます。」
3	仮説 質問	分析仮説は「システムのデフォルトポートが解放されていないためか？」です。	
4	境界 質問	条件を明確に定義し、 シナリオ	「WindowsではJettyを使用すると8080 ポートが原因で..."
5	検証 質問	テスト可能かつ再現可能 最新かつ検証可能	「ランダムポートに構成を変更した後、システムを正常に起動できますか？」 動く？"

—

最初のレベルはあいまいです。

第5段階は確信です。—

歩ずつ質問するたびに、理解がより明確になります。

「繰り返し質問法」を使用します。

質問をするときは、常に次のことを自問してください。

1.「この質問は明確に尋ねましたか？」

2.「どんな文脈を見逃しているのか？」

3.「より小さく、より具体的なサブ問題は存在しますか？」

4.「この問題は検証できますか ?再現できますか？」

5.「私は『症状』について尋ねているのか、それとも『根本原因』について尋ねているのか？」

—

これは実際には一種の「認知デバッグ」です。

要約:

多くの人は、質問するだけで答えが得られると考えています。しかし、真実は、真の

質問は曖昧さを解消するために何度も質問する必要があるということです。

最初の質問は、暗闇を照らすマッチにすぎません。

2 番目の質問は、表のアウトラインを確認することです。

3番目の質問をした後で、私は勇気を出してドアに向かって歩き始めました。

継続的に質問することは、混沌から論理的な境界を磨くことであり、あなたと世界との間の「共同モデリング」のプロセスです。

—

提案：

問いは投げるダーツではなく、踏み出す一歩です。一歩踏み出すごとに、世界

はより鮮明になります。

第5章 情報の検索方法 :質問を使って検索を導く

---

1. 情報を探すときに最も誤解されやすいことは何ですか?

多くの人は、良い質問をしても、良い答えを見つけることができません。その理由は次のとおりです。

間違った行動	なぜ非効率的なのでしょうか?
「自然言語」を使ってGoogleで検索する / チャットGPT	あいまいなキーワードのため、システムはコアを見つけることができません

エラーを直接コピーして検索すると、自分のコンテキストではなく他の人の問題が見つかる可能性があります。

「最終的な答え」を直接知りたい	探索とは、答えを運ぶことではなく、認知を構築することである
文書を読まずに、ブログ、TikTok、Xiaohongを検索する	私たちは本来の意味を理解できなくなり、「言い換えられた」ものしか得られない。
本	断片

—

検索とは「答えを探すこと」ではなく、「認知モデルを構築するプロセス」です。

---

2. 情報を見つけるための4段階アプローチ :RATEフレームワーク

ステップ	意味	アクション
R – 精製 キーワードに焦点を当てる	「問題の説明」から核となる単語を抽出する	例 :Spring + YAML + リロード + アクチュエータ
A – 自分自身を明確にするために質問する 何を探すべきか	設定手順をお探しですか ?使用例をお探しですか ?エラー報告 分析しますか?	質問の種類を記入してください
T – ターゲット プラットフォーム/ドキュメント	対応する使用法: 公式ドキュメント/スタック オーバーフロー / GitHubの問題 / ChatGPT / MDN / など	サイト:spring.io + 「リロード」

ステップ	意味	アクション
E – 評価 認証情報の信頼性と適用性 ユーザビリティ	過去2年以内ですか ?あなたにも当てはまりますか？ 問題は再現可能でしょうか?	環境/バージョン/言語などを確認してください。 一貫性がない

—

答えを見つけるだけでは意味がありません。「正しいかどうか」「適切かどうか」「なぜ正しいのか」を判断できる必要があります。

---

3. 情報検索のための一般的な戦略

コンテキスト検索戦略		ツールの推奨事項
コンセプトを確認する /使用法	キーワード + 「公式ドキュメント」	サイト:docs.python.org、MDN、man7.org、 等
エラーを確認 情報	エラーキーワード + プロジェクト名 + ボックス フレーム名 + バージョン	Google + スタックオーバーフロー
構成を確認する パラメータ	プロジェクト名 + パラメータ名	site:spring.io + “server.port”
セマンティクスを確認する 違い	「xxx vs yyy」	「Javaにおけるリストとセット」
類似品をチェック 経験	GitHub の問題 / ChatGPT / ブログ + 独自バージョン環境	「Redis タイムアウト スプリングブート サイト:stackoverflow.com

—

ユーザーにコツを教える: ブログやコンテンツの氾濫を避けるために、検索ドメインを制限するには、site:xxx.com を追加します

農場。

---

4. 情報を見つけたら、すぐに使用せず、検証してください。

見つけたコンテンツはすべて検証する必要があります。

1. どのバージョンに適用されますか ?環境は一貫していますか？

2. 隠れた副作用はありますか?

3. どのような前提があるのか ?それを満たしているか？

4. まだ見つけていない必要な情報はありますか？（例 :設定順序、依存関係のアノテーション）

—

このステップは「最小限の再現可能な実験環境を構築する」と組み合わせることができます。

検証については以下のセクションを参照してください。

第6章 答えを見つけることから理解を確認することへ :知識を確信に変えるプロセス

認知を最後の重要な段階まで押し進めました。

答えを見つけたからといって、それが「理解した」あるいは「問題を解決した」ということではありません。

真の「自信」は、自分自身で検証し、それが特定の状況で本当に機能することを確認することから生まれます。

効果。

---

1. 検証はなぜ必要なのですか?

情報自体が偏っていたり、適用できない可能性があるので:

情報源の潜在的な問題

ChatGPT は高い確率での説明を提供しますが、コンテキストと一致しない可能性があります。

検索結果は、古いバージョン、異なる環境、または誤解されたコンテキストである可能性があります

情報源の潜在的な問題

ブログ投稿には著者が気づいていない偏見や制約が含まれている可能性がある

公式ドキュメントは曖昧であったり、動作を理解するために複数のポイントを組み合わせる必要がある場合があります。

他の人の経験は彼らには当てはまりますが、あなたには当てはまりません。

—

したがって、次のことを自分自身で訓練する必要があります。

情報を受け入れるのではなく、検証された理解のみを受け入れます。

---

2. 検証の中核目的

情報を見つけたら、次の 3 つの質問に答える必要があります。

1. 現在の環境で効果的ですか?
2. 期待通りに動作しますか ?副作用はありますか ?
3. なぜそれが機能するのか、または機能しないのか理解していますか?

—

一言でまとめると：

実行可能

説明可能

移行可能

これが「知識を確信している」という意味です。

---

3. 5段階認証方式 :VALIDモデル

ステップ	意味	操作する
V – 環境1の検証	バージョン、言語、フレームを確認する	例えば、Python 3.11と3.6の違い
に	フレームワークとシステムの一貫性	異、Spring Boot 2 vs 3
A – 適用する 最も多くを構築する	サンドボックス環境でソリューションを再現する	一時的なプロジェクトまたはテストクラスを構築してみる
小さなテストケース	解決	
L – ログ比較動作	中間ステータスをログに記録して出力する	期待通りに作用しますか ?副作用はありますか？
そして期待	状態、観察の違い	生まれる？
I – 分離	一度に検証される変数は 1 つだけです。	「たくさん変えて、どれから始めればいいのかわからなくなる」という状況を避ける
変数	構成項目	使用"
D – 文書	現象を要約する → 推論エンジン	このパラメータはxxxの後に設定する必要があります
結果を確認する	→ 結果ノート	そうしないと効力を発揮しません。」

—

考え方:

検証されていない「答え」は、単に他人の推測にすぎません。

検証された知識はあなたの理解です。

IV. 検証方法：一般的な種類と方法

シナリオ	検証方法
パラメータ設定が有効かどうかを確認します。設定項目を変更 → サービスを再起動 → ログ/動作の変化を確認します。	
関数呼び出しが期待どおりに中断されたかどうかを確認し、入力および出力パラメータを出力し、パスをチェックします。	
Postman/curl/devtools を使用してリクエストを送信し、レスポンス本文を観察して、ページが正しく応答しているかどうかを確認します。	
権限制御は効果的ですか ?異なるユーザーロールでアクセスしてみてください	
例外はキャッチされましたか? 例外シナリオを作成し、ログ出力が期待どおりかどうかを確認します。	

シナリオ	検証方法
新しいロジックが古いロジックをカバーしているかどうか。ユニットテストツールやカバレッジツールを使用して、パスが成功しているかどうかを分析します。	—

AIによる提案を活用したい場合は、自分でも「最低限の検証環境」を構築する必要があります。

検証環境とは、たとえミスをしても損害が出ない環境のことです。これは非常に重要です。

たとえば、実験によってデータやドキュメントが削除される可能性がある場合は、これらのアクションによって損害が発生しないことを確認する必要があります。

最悪の事態でもデータやドキュメントを復元できるように、バックアップを作成する必要があります。

推奨されるセクションタイトル:

- 「情報の確認は始まりに過ぎず、検証は終わりです」
- 「知識を実践するための最終ステップ：個人検証」
- 「覚えておいてください：検証がなければ、理解は幻想です」

推薦する：

他の人が「はい」と言うのを見るだけでは意味がありません。

「試してみました。うまくいきました。理由もわかりました」と言えるようにする必要があります。

第7章 認知の明確化は線ではなく循環である

質問、情報、検証の三重らせん構造

---

1. 線形認知の錯覚：多くの人がそう考えている

プロセスは次のようになります:

1. 質問する

2. 情報を確認する

3. 答えを得る

4. 応募する → 完了！

これは幻想です。この「入出力」の線形思考は標準化の問題には適していますが、実際には標準化の問題には適していません。

開発、設計、デバッグ、学習のプロセスにおいて、ほぼすべてが失敗しました。

—

現実は次のようになります。

最初の質問 → 確認 → ある程度の情報を得る → しかし、理解が

不十分だと気づく → 質問を修正し直す → もう一度確認 → もう一

度検証 → またダメ → 質問文をもう一度修正 → … → 最終的に、積み重ねに頼る

自信を築きましょう。

これが本当の認知プロセスです。

—

2. 認知の三重らせんモデル：PQVサイクル

このモデルを読者に紹介することができます:

ステップ	意味	行動
P -	問題質問、定義が不明瞭な場合に理解できない点を明確にする	
Q -	クエリは情報を見つけ、説明を構築し、ドキュメント/AI/検索を使用して候補の回答を見つけます	
V -	検証実験検証、行動の観察、期待と現実の比較により認知の正確性を確認する	

このサイクルは一度きりのものではありませんが、

•各サークルごとに理解が深まります。

失敗するたびに「P に戻って」問題を再定義する必要が生じます。

•最終的には、「一度で正しい答えを見つける」ことに頼るのではなく、「真実に近づくために複数回のサイクル」に頼ることになります。

—

それはネジをドリルで穴を開けるようなものです。

ターンごとに同じ場所に留まらず、どんどん奥深くへ進んでいきます。

—

説明例:

例えば：

1. 質問：「システムログイン後にリダイレクトが失敗するのはなぜですか？」

2. セッションが失われたという情報があるため、Cookie 設定を確認することをお勧めします。

3. 試してみたが、設定が無効であることが判明しました。

4. その後、元の質問が範囲が広すぎるため、「なぜ sessionId が保存されないのですか?」とすべきであることに気付きます。

所有？"

5. 再度検索する際に、nginx、cross-domain、set-cookie などのキーワードを追加しました。

6. 最終的に、パスの混乱はプロキシ構成ストリップパスによって発生したことが判明しました。

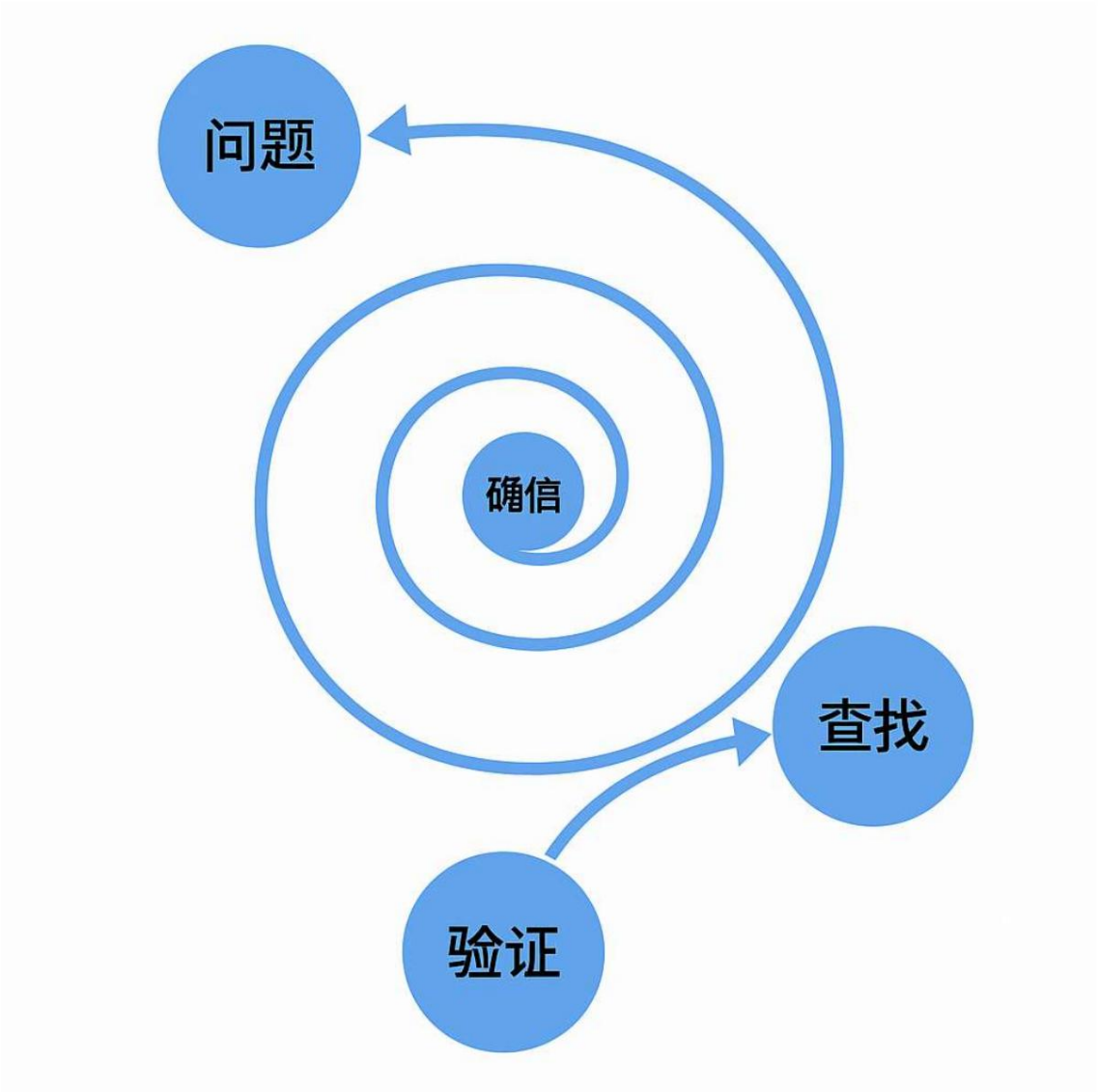
7. この時点で、Cookie、セッション、およびプロキシの動作全体を体系的に理解できるようになります。

—

このプロセス全体には、単に「確認して完了」するのではなく、3 つのステップを繰り返し実行することが含まれます。

—

図の提案 :認知サイクル図（私が描きます）



段落の要約:

知識は一度の質問で得られるものではなく、繰り返し磨くことによってのみ得られ

ます。確実性は答えではなく、継続的な検証の後に残る核心です。

戻って再度質問する気があるなら、あなたはすでに明確さへの道を歩んでいることになります。

第8章 自信過剰に注意

—理解したと思っても、それは単なる出発点に過ぎない

私たちは拡大鏡を使って、曖昧さから確実性への認知の旅を徐々に解体してきました。

曖昧さはあなたのせいではないこと、そして確信は一夜にして得られるものではないことに気づき始めるかもしれません。

しかし、次のパラドックスに注意してください。

「曖昧さから確実性へ」についてのあなたの理解は、現時点ではまだ曖昧です。この資料を読んだからといって、それを完全に

理解したわけではありません。

それはあなたの旅の始まりに過ぎません。

これらの概念に対する理解はまだ断片的で曖昧であり、読む前よりも混乱している可能性もあります。

混乱している — これは正常なことで

す。なぜなら、あなたは「認知障害」の期間を経験したばかりだからです。古いモデルは崩壊し、新しい構造はまだ確立されていません。

それで：

「すべて理解する」ことを急がず、「少し理解で

きた」部分だけをピックアップして、実際に試してみましょう。そして戻ってきて、もう一度読み、理解

を深め、認知マップを修正してください。

完全に理解する必要はありません。ただ「使い始める」だけです。

実践していくうちに、漠然とした思いから少しずつ確信が育まれていきます。

—

ある日、あなたはもはや曖昧さを恐れず、着実に問題を把握し、曖昧さからモデルを構築できることに気づきます。

解決策を構築して初めて、その本を本当に「学んだ」ことになります。

その時までに、あなたのプログラミング能力、問題解決能力、そしてチームコミュニケーション能力は新たなレベルに飛躍しているはずです。

レベル。

しかし、覚えておいてください。これは終わりではありません。

現時点でのあなたの確信はすべて、将来的には新たな漠然とした出発点となるでしょう。

古い認識を絶えず解体し、新しい理解を確立することが、「意識的なプログラマー」としての私たちの生き方です。

—

それで：

- この本のフレームワークを繰り返し再利用します。
- この考えを同僚や後輩に伝えてください。

「これはもうわかっている」という考え方には注意してください。

なぜなら：

「本当の危険はぼやけた状態ではなく、はっきりと見ているという思い込みである。」

「時代の変化に対応することは選択肢ではなく、生き残るための方法だ。」

幸運を！

最後に書いた

『心の壁を取り壊す』という本があります。

『心の壁を取り壊せ』は中国で大きな影響力を持つ認知科学のベストセラーです。

強調して書く：

人々の問題の多くは「自分自身が築き上げた精神的な限界」から生じており、「壁を壊す」ことは、固有の認識を破壊し、自分と世界との関係性を再定義するプロセスです。

「曖昧さから確実性へ」の共鳴点:

心の壁を壊す　　曖昧さから確実性へ

焦点は、心理的な認知障壁とプログラマーの認知的曖昧さにあります。

心の壁を壊す　　曖昧さから確実性へ

「壁」とは、思考における目に見えない限界です。「ぼやけ」とは、問題の境界が不明瞭なことです。

本来の信念を振り返ることを提唱する　　誤った信念や幻想を見抜くことを主張する

典型的な論調は心理的ヒューリスティックスです。典型的な論調は認知システム工学です。

どちらも、人間が次の 1 つのことを行うのを支援します。

「自分の脳に騙される」状態から抜け出しましょう。

心の壁を壊す接続ポイント:

「人々が失敗する理由は、何かをする方法を知らないからではなく、それをもう一度理解する気がないためです。」

ぼやけた影の下には思考の壁が横たわっている。

そして自信とは、壁が取り壊された後のビジョンなのです。」

警告：

- 自信過剰になると、情報にアクセスすることが難しくなるだけでなく、その情報源を疑問視することもできなくなります。
- それはあなたを愚かにするわけではありませんが、「あなたの知性の古いバージョンだけを信頼する」ようになります。

それは暗闇ではなく、「あなたがいつもそう思っていた光」なので、あなたはそれを疑うことはなかったのです。「過剰な確信

信仰は思考の壁の中で最も強いレンガです。」

- 「信じることから再理解へ」

「経験は壁ではなく、解体して再構築すべき足場です。」