システム開発フロー

要件定義
（ゴール設定・達成条件）

基本設計
・主要機能の列挙
・データベース構造
・データフロー[IN-OUTレベル]

詳細設計
・処理ロジック
・データフロー[詳細]
・データスキーマ（形式と表現）
・利用ライブラリ選定

実装（コーディング）

テスト
・単体テスト（関数・クラス単位）
・結合テスト（プロセス単位）
・全体テスト

生成AIを使った効率化

基本的にたたき台と修正までにした方が良く、全任せはこれじゃ無い感があること多い。
但し、全く知見が無ければ、作り直す前提で全任せして感覚をつかむのは大有り。

AIにさせること

・たたき台の作成
・修正指示と清書

・要件から機能に分解
・使うデータの列とテーブル構成
・データのIN-OUTの区切り
・"基本設計"が要件を満たすかチェックさせる

詳細設計
・データの具体的な処理プロセスを提案させる
・処理に必要なライブラリを提案させる
・データのスキーマを提案させる

実装
・実装内容を提案させる
・関数化、クラス化を提案させる
・システム全体のファイル・ディレクトリ構成を提案させる

・テスト用ファイルを作成
・テスト用のサンプルデータ作成
・エラーが出たら修正

ポイント

・指示に、「最低限の要件で」などと言うと整理しやすい：つけないとメイン要件以外もモリモリで出力するので、どこが基本要件なのか分かりにくくなる。セキュリティやエラー処理などは後々に膨らませればよい。
・ここでシステム全体のIN-OUTをイメージする。確定出なくても説明可能になれば次に進んでみる。これじゃ無い感がでれば戻ればよい。

・データベース構造は特に仕様モリモリで提案される傾向に感じる。最終出力データを意識して、それを出力するために必要最小限のINPUTや加工に必要な追加INデータを集めるイメージ（プロセスのメインデータ）。
・「データ処理を管理するためのデータ」が必要になった時追加するイメージ（処理中のステータスやデータを一意に識別するためのID、時刻、ユーザー、権限など）
・人間が要件と基本設計の関係性を理解するのがとても大切。基本設計を丸投げするのは結果的に手戻りになるので、基本設計のロジックは全部理解すべき。そのためにAIに聞きまくる（なぜこれを選んだのか、このデータは何に使うのか、とか）
・コーディングAIでは処理理論のWebサーチができない（教師データ固定）ので、AIチャットツールを別で使える方が良い。特にディープリサーチは超有用。英語文献も含めて世界レベルで調査＆提案してくれる。

・要件⇔基本設計とは変わって、全部／ほぼお任せして良い。処理のINとOUT（基本設計）がしっかりしていれば、その中間の処理は何でもよい。処理効率性、シンプルさ・コード文量（読みやすさ＝保守性）、API利用コストとかで適切に記載され、動作すればとりあえずヨシにできる（後で見るけど、システムの効果確認を優先できる）
・データ処理のイメージができるなら、データ処理フローは自分で設計してスキーマを指定してあげると、横展発想がしやすいのでお勧め。
・エラー回避処理をモリモリ作るツールやモデルがあるので、メイン機能が固まるまでは「最低限の処理ができるように実装」などと指定した方が良い。のちのテスト実行時にエラー出たが、エラー回避処理で問題なしになることがあった。
・部分的実装、全体実装ができたらAIに実装内容をドキュメントとして作成させ、それを人間が読んで流れを把握すると楽。ドキュメントは修正や要件・基本設計との照合、テストファイル作成など利用価値高い。

・ログ出力の実装（print() or ファイル出力･･･ファイル出力がおすすめ）
・エラー or 意図せぬ処理があったらログを提示して原因調査と修正を指示する（ログをファイル出力するとパス指定するだけなので楽）
・要件定義や基本設計でドキュメントをしっかり作っておくと、テストの齟齬もなくスムーズ