

# AI Assisting Software development

## Keep the Human in the Loop Software Development

This is the detailed breakdown of the optimized 7-step **Human-in-the-Loop (HITL)** software development lifecycle. This structure balances the speed of AI with the necessity of human oversight and automated safety.

unnamed.jpg

### 1. Requirements & Context Gathering

**Description:** Gathering user stories, technical constraints, and existing codebase knowledge to form a "context packet" for the AI.

- **The Reality:** **This is the single most important step.** If you fail to provide the AI with your specific coding standards or database schema, the output will be generic and incompatible.
- **The Risk: "Domain Blindness."** The AI assumes standard practices (e.g., SQL) even if your project uses a specific variation (e.g., NoSQL) because it wasn't explicitly told.
- **Optimization:** Use a **RAG (Retrieval-Augmented Generation)** system. Instead of pasting code, point the AI to your repository so it "reads" your existing architecture before suggesting new features.

#### Prompt Template

```
# ROLE DEFINITION
```

```
You are a Senior Software Architect and Lead Developer. Your goal is to assist in planning and implementing a new feature while adhering strictly to our existing technology stack, coding standards, and architectural patterns.
```

## # PROJECT DNA (CONTEXT)

- \* **Project Name:** [e.g., FinTrack Pro]
- \* **Core Functionality:** [e.g., A SaaS platform for small business accounting]
- \* **Tech Stack (Backend):** [e.g., Node.js, Express, PostgreSQL, Prisma ORM]
- \* **Tech Stack (Frontend):** [e.g., Next.js 14 (App Router), Tailwind CSS, TypeScript]
- \* **External Services:** [e.g., AWS S3 for storage, Stripe for payments]

## # CODING STANDARDS & CONSTRAINTS

- Style:** [e.g., Functional programming preferred. Use const/let, no var. Strict TypeScript types—no 'any'.]
- Testing:** [e.g., Jest for Unit tests. React Testing Library for components.]
- Security:** [e.g., Never hardcode secrets. Use environment variables. Validate all inputs using Zod.]
- Architecture:** [e.g., Repository pattern for database access. Service layer for business logic.]

## # THE CURRENT TASK (REQUIREMENTS)

I need to implement the following feature:

- Feature Name:** [e.g., User Profile Image Upload]
- User Story:** [e.g., As a user, I want to upload a profile picture (max 5MB, JPEG/PNG) so that I can personalize my account.]
- Specific Constraints:** [e.g., Images must be resized to 500x500 before storage. Store metadata in the 'users' table.]

## # EXISTING CODE CONTEXT

(Paste relevant snippets of existing code here, such as your `schema.prisma`, a related API route, or a similar component to mimic style. If using an IDE with context, reference the files: @user.ts, @upload-service.ts)

## # INSTRUCTIONS FOR STEP 1 (ANALYSIS ONLY)

**DO NOT WRITE CODE YET.**

Please analyze the requirements and my existing context.

- Restate the requirements in your own words to prove you understand.
- Identify any missing information or potential risks in the requirements.
- List the specific files you expect to modify or create.

Wait for my confirmation before proceeding to the Implementation Plan.

## 2. AI Assisted Planning

**Description:** The AI proposes an implementation strategy, file structure changes, and library choices before writing code.

- **The Reality:** AI is an excellent "middle manager." It is great at breaking a complex feature into 5 or 6 distinct sub-tasks.
- **The Risk: "Hallucinated Dependencies."** AI often suggests libraries that are deprecated, don't exist, or are overkill for the task.
- **Optimization:** The human must act as the **System Architect**. Do not approve the plan unless you recognize the libraries and agree with the data flow.

## 3. AI Code Generation

**Description:** The AI generates the actual code based on the approved plan, typically in iterative chunks.

- **The Reality:** This provides massive velocity, especially for boilerplate, API integrations, and standard algorithms.
- **The Risk: "Context Drift."** As the AI moves from modifying File A to File B, it may forget the changes it just made, leading to mismatched variable names or import errors.
- **Optimization:** Use **Step-by-Step Generation**. Don't ask for the whole app at once. Ask for the database model, then the API layer, then the frontend, verifying each chunk briefly.

## 4. Automated Security & Linting (The Guardrail)

**Description:** A strictly automated step where code is run through static analysis tools (SAST) and linters before a human looks at it.

- **The Reality:** AI code is often messy or utilizes insecure patterns (e.g., hardcoding API keys or using `eval()`).
- **The Risk: "Supply Chain Attacks."** AI has been known to hallucinate package names that attackers have registered with malicious code.
- **Optimization:** Configure your CI/CD (continuous integration, continuous deployment) pipeline to **auto-reject** AI code that doesn't pass standard linting or security checks. Do not waste human time reviewing code that doesn't compile or looks messy.

## 5. AI Assisted Testing

**Description:** The AI analyzes the code it just wrote (and the requirements) to generate Unit Tests and Integration Tests.

- **The Reality:** AI is surprisingly good at finding edge cases humans miss (e.g., "What happens if the input is null?").
- **The Risk: "Tautological Testing."** The AI often writes a test that confirms its own buggy logic (e.g., "If function returns Error, expect Error" - passing the test, but failing the requirement).
- **Optimization:** Instruct the AI to write tests based on the **Requirements (Step 1)**, not the **Code (Step 3)**. Ideally, use Test Driven Development (TDD) where AI writes tests *before* the code.

## 6. Human Evaluation & Refactoring

**Description:** The human developer reviews the logic, runs the app, and refactors for readability and maintainability.

- **The Reality:** This is the hardest step. Reading code is more cognitively demanding than writing it.
- **The Risk: "Review Fatigue."** Humans tend to trust the AI after the first few correct lines and skim the rest, missing subtle logic bombs or business logic errors.
- **Optimization:** Focus the human review on "**The Why**" rather than "The How." Does this solve the user problem? Is the UX good? Let the AI handle syntax; let the human handle value.

## 7. Issue Detection & Continuous Improvement

**Description:** Monitoring the application in production and using AI to parse logs/errors to suggest fixes.

- **The Reality:** AI can correlate error logs across distributed systems much faster than humans.
- **The Risk: "Alert Noise."** Without proper tuning, AI might flag standard warnings as critical errors, desensitizing the team.
- **Optimization: The Feedback Loop.** When a bug is found and fixed, that specific case must be added to the "Context" of Step 1 for future features. "Remember, we previously had an issue with X, so ensure Y is handled."

---

Revision #3

Created 2025-11-24 19:06:01 UTC by Max

Updated 2025-11-27 20:45:49 UTC by Max