Problem Standpoint & Framing

I initially approached the problem from my industry experience as a frontend-focused software engineer for the past 5 years.

How might we go from design to technical implementation more efficiently, with less cognitive labor, while recognizing the iterative (and often improvisational) nature of the development process?

Market Research

After extensive market research, I discovered 3 major categories that aim to address the pain at this interface between design and development, offering additional insight into the problem space and the limitations of current solutions.

1. Visual App/Site Builder (10 examples)
2. Design-to-Code (12 examples)
3. AI Code Suggestions (1 example)

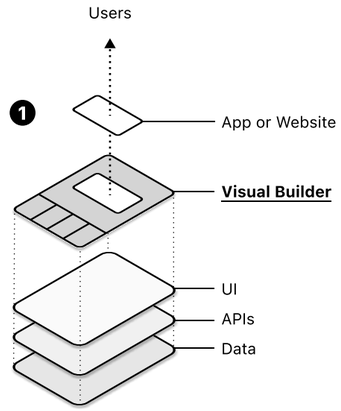
While I initially focused on visual builders and design-to-code solutions, I quickly realized that they're limited in scope compared to the highly expressive, multilayered reality of software systems.

AI-powered code suggestions are an unconventional approach to improving the developer experience, broadly useful to developers working in any context. Due to the dependence on "hard tech", this solution is currently limited to a single industry contender: GitHub's Copilot.

Insights & Opportunities

Copilot flips the problem on its head, honoring natural developer workflows without forcing anyone into a rigid structure or complicated toolchain. The AI is constantly trained by developers across a variety of contexts and domains–similar to the training of self-driving cars.

How might we upskill the AI to move beyond isolated functions to instead perform end-to-end feature development autonomously, with human supervision?



Intended User

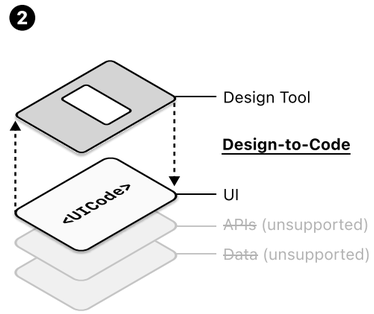
- Individuals & Small Teams

Strengths & Uses

- Accessible to non-technical contributors
- Low-code, build visually
- Create and publish fully functional apps
- Prototypes, MVPs
- Marketing Sites
- Internal Tools

Limitations

- Does not fit with technical requirements of scaling business contexts
- Lack of customization across the technical stack
- Locked into basic UI primitives
- Significant manual labor



Intended User

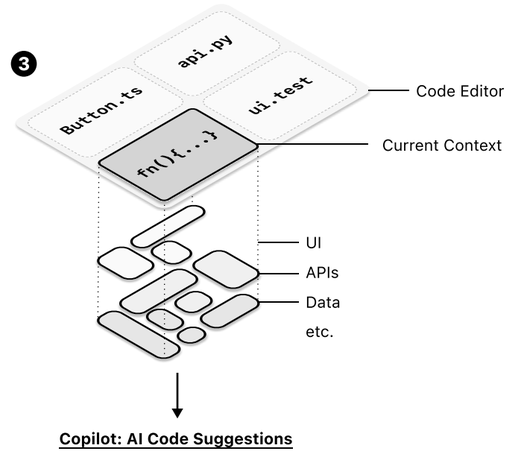
- Designers & Developers

Strengths & Uses

- Reduces translation labor for developers
- Maintain a single-source of truth–production matches design
- AI-powered solutions automatically infer component hierarchy and interactivity

Limitations

- Generated code is decontextualized from local dialects across codebases
- Fragile, sophisticated toolchain
- Only supports presentational aspects–no application behavior



Intended User

- Developers

Strengths & Uses

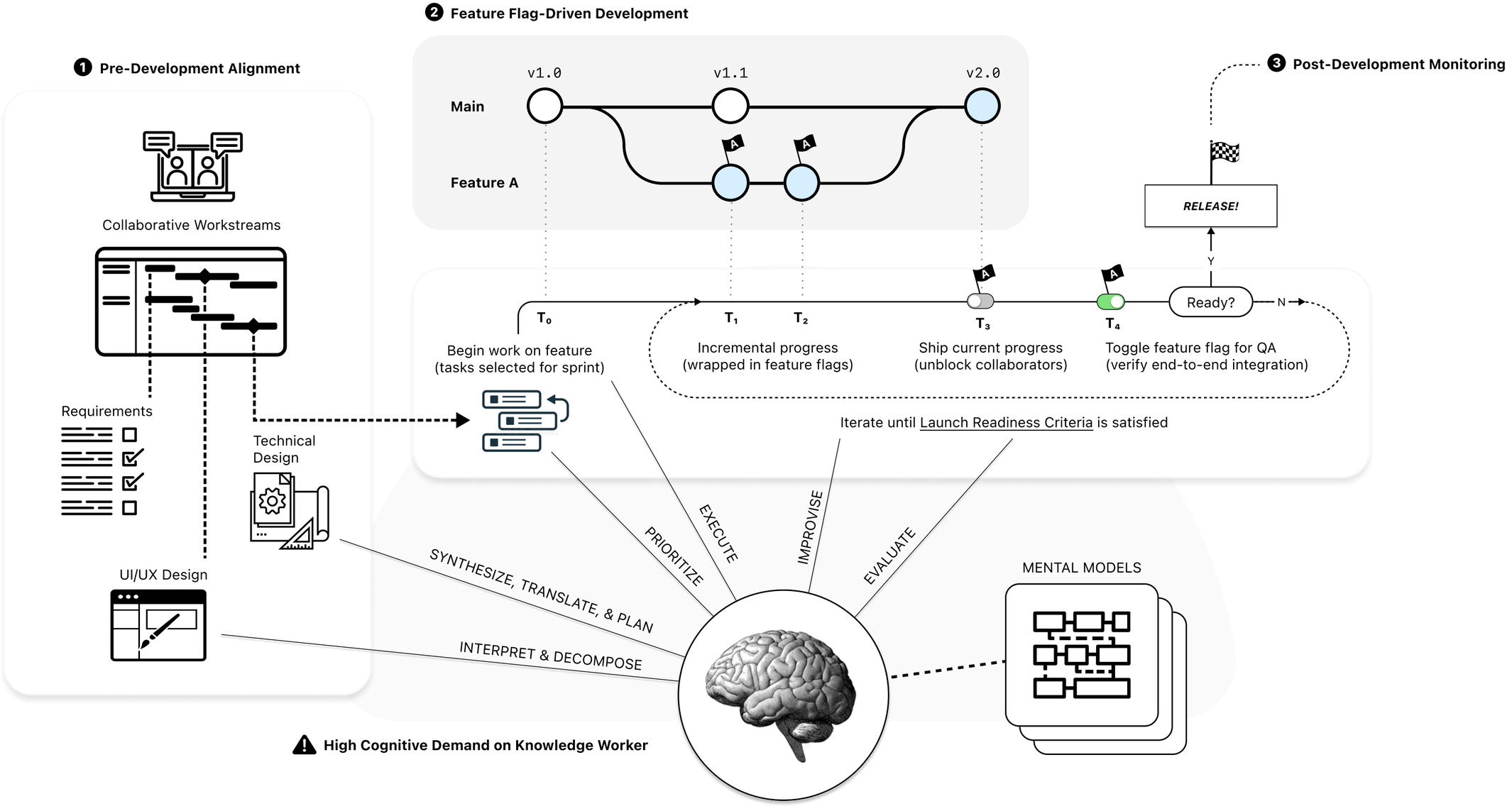
- Highly expressive–AI code suggestions are contextually relevant
- Conventional, familiar interaction pattern
- Reduces cognitive labor for focused tasks–keeps developers in flow
- Works across the entire stack
- Improves with use over time

Limitations

- Must provide instructions as code-comments or function signature to trigger suggestions
- Isolated to function-level micro-tasks–unaware of the higher level developer intent or task context
- Unaware of other tools and artifacts–we still must manually translate design to code since the AI cannot "see" the design or requirements

Contextualizing Software Development

Knowledge workers typically emit artifacts such as schedules, plans, priorities, designs, and communications while collaborating across all phases of development. Rules, norms, and industry standards structure the way the work gets down and how the product gets built. All of this is juggled by the modern knowledge-worker as a sort of high level supervisory activity that guides their actions. To make our AI-powered software development assistant more useful, it will need awareness of these activities, and access to the knowledge distributed across these artifacts.



Role-Reversal

GitHub's Copilot is valuable insofar as it helps to shortcut some extra lines of code as you work–it disappears into the background until it can make itself useful. But you're still in charge. That means you're still holding all the knowledge and references in your head as you navigate a complex codebase to build whatever it is you're building. What if the AI knew what it was you were planning to build from the start, and could build it out before your eyes?

That's exactly the type of role-reversal we should borrow from Tesla's Autopilot, where you're in the driver's seat, but mostly tagging along for the ride as you go from A to B. And since writing code isn't bound by time or space in the same way as driving, applying this technology to the software development context not only frees up cognitive space for knowledge workers, but may also improve development velocity by an order of magnitude.

Copilot GitHub OpenAI

sentiment.ts

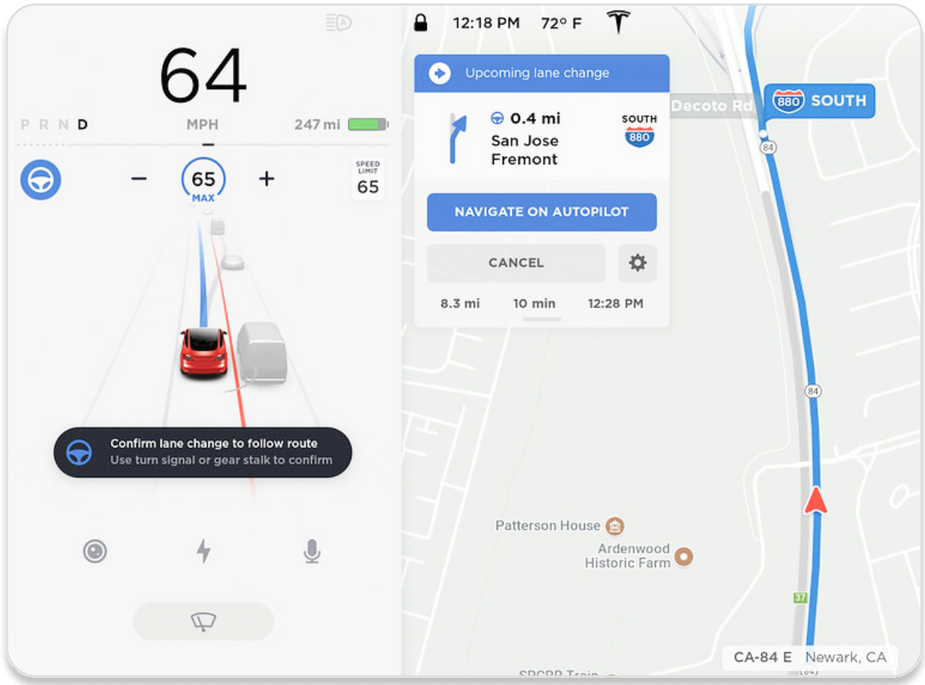
Next ⌥] | Previous ⌥[| Accept Tab

```
#!/usr/bin/env ts-node

import { fetch } from "fetch-h2";

// Determine whether the sentiment of text is positive
// Use a web service
async function isPositive(text: string): Promise<boolean> {
  const response = await fetch(`http://text-processing.com/api/sentiment/`, {
    method: "POST",
    body: `text=${text}`,
    headers: {
      "Content-Type": "application/x-www-form-urlencoded",
    },
  });
  const json = await response.json();
  return json.label === "pos";
}
```

Copilot

GitHub's AI pair programming assistant

Autopilot TESLA



Tesla's self-driving interface

Isomorphic Problem Structure

Autopilot
(Self-Writing Software)

"Let's build feature X"

Environmental Observation

- Product requirement documents
- UI/UX design artifacts
- Technical design documents
- Project communications (e.g. Slack channels)
- Project workstream (backlog, sprint tasks)
- Code repositories
- Code editor context
- Focused/background applications/tabs
- Interactive UI scenarios (app/web context)
- Anonymized, sanitized telemetry

Informs

Helps to clarify the criteria for determining when the goal state has been reached

Passively detect developer intent

Detect invalid assumptions about the current system

Planning & Instructions (A → B)

Goal State

- Current system behavior
- New feature/system behavior
- Prioritized tasks (acknowledges dependencies)
- Pivot/iterate/scope-change as needed

Inform

Policies (Rules & Norms)

Determine / Govern

- Code compiles and runs
- UI matches design intent
- Complies with lint rules
- Complies with code coverage thresholds
- All unit/integration tests pass
- Complies with global governance standards/policies
- Follows local conventions
- Ontologically relevant code
- No insecure code
- No unsafe (or harmful) code
- Wrapped with feature flags (incremental ship, avoid bugs)
- Meets non-functional requirements
- Require human approval before merging code
- Limit/sandbox allowable terminal commands

Higher Order Actions (Ensemble)

(Not exhaustive)

- Create/modify UI components (based on design artifact)
- Modify app navigation
- Modify database schema
- Define/modify data models
- Define/modify/expose APIs/endpoints
- Structure and manage local state
- Implement business logic for expected behavior
- Make use of available dependencies
- Write/modify unit/integration tests

Composed of

Low Level Actions (Input Controls)

- Accept/cycle through code suggestions
- Manually modify code (human input)
- Run git commands (e.g. branch, commit, PR, etc)
- Run unit/integration tests
- Start/stop a service/application
- Execute commands/jobs exposed by configuration

Action-feedback loop for validating code changes with human supervision

Autopilot
(Self-Driving Car)

"Take me to the office"

Environmental Observation

- Sensors (GPS/LiDAR/video)
- Real-time traffic data/alerts
- Street maps (w/ metadata)

Informs

Planning & Instructions

Goal State

- Current location
- Destination address/geolocation
- Directions (rerouting as needed)

Inform

Policies (Rules & Norms)

Determine / Govern

- Obey state driving rules
- Obey speed limits
- Obey local signage
- Improvise to match traffic
- Avoid safety incidents
- Require human confirmation for certain actions

Higher Order Actions (Ensemble)

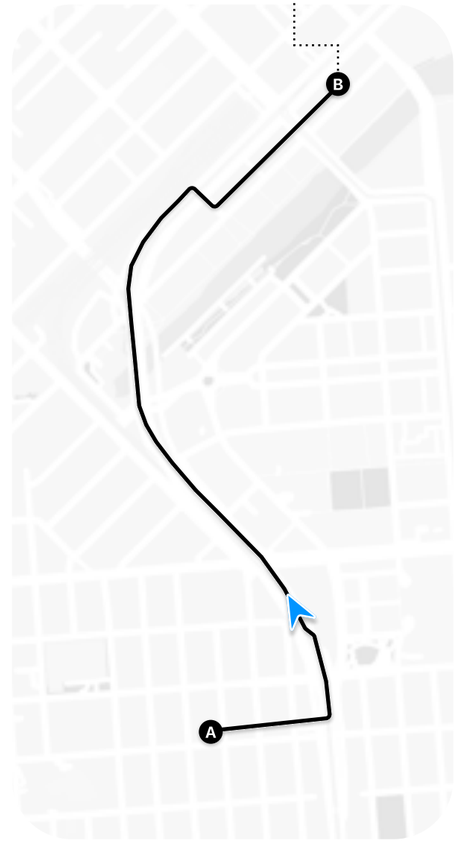
(Not exhaustive)

- Navigate intersection
- Merge onto freeway
- Flow with traffic
- Change lanes
- Exit freeway
- Turn onto street
- Proceed along route
- Cross train tracks
- Navigate obstacle
- Pullover/Park

Composed of

Low Level Actions (Input Controls)

- Accelerate
- Brake
- Signal for turn/stop
- Steer
- Shift to Park/Reverse/Neutral/Drive
- Toggle exterior lights



While these problem domains appear completely unrelated on the surface, ***the underlying problem structures are highly isomorphic****—we can look at software development activities as wayfinding and terrain traversal across a technical substrate akin to self-driving navigation across roadways in the built environment.*