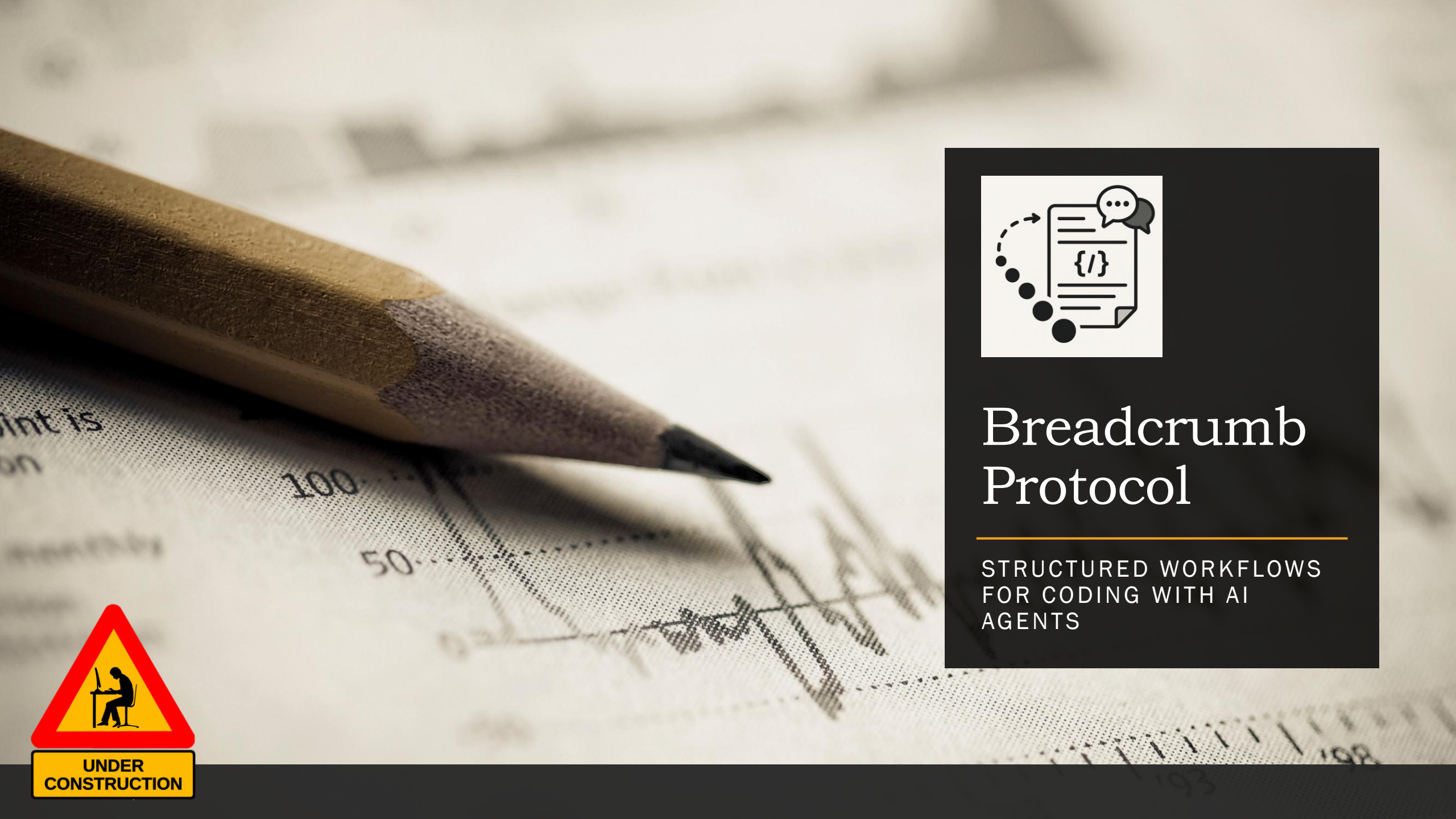
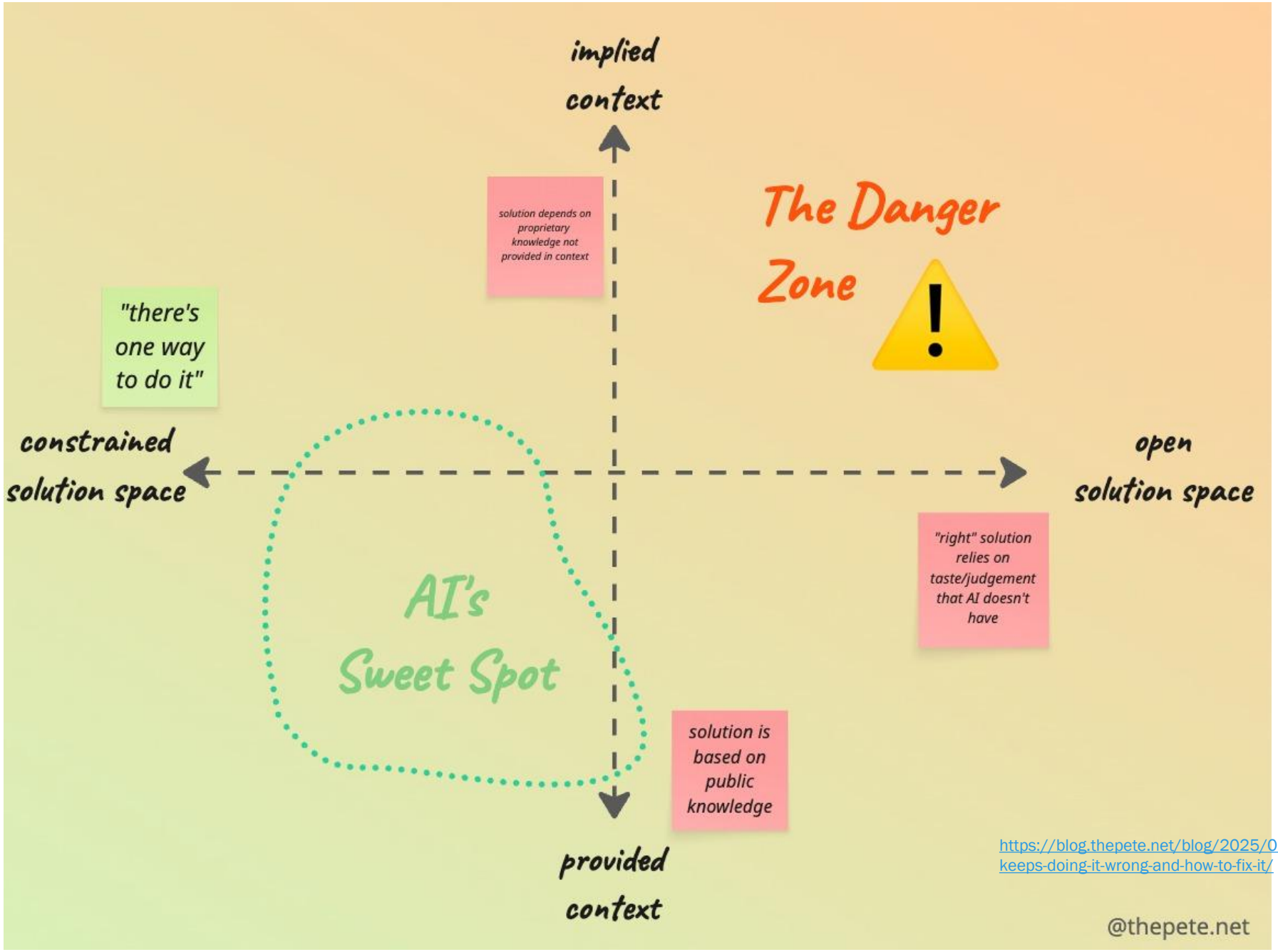


Breadcrumb Protocol

STRUCTURED WORKFLOWS FOR CODING WITH AI AGENTS



https://blog.thepete.net/blog/2025/05/22/why-your-ai-coding-assistant-keeps-doing-it-wrong-and-how-to-fix-it/

Problems when coding with AI

- AI suggestions may contradict established patterns or architectural choices specific to solution.
- Critical decisions and their rationale remain trapped in ephemeral conversations.
- As context grows, it's harder to ground the agent on what's important.
- No easy way to amend the plan of the agent midway through a session.
  - It's hard to see what the agent sees and thinks.
  - Hard to rewind to a point and start from there.
- Inconsistent experience across the team.

Breadcrumb Protocol Core Tenets

The breadcrumb is a dynamic document that evolves as you and the agent go deeper into a solution. Acting as a scratch pad of your current understanding.

Increase feedback surface to the agent

Shorter iterations

Focussed context

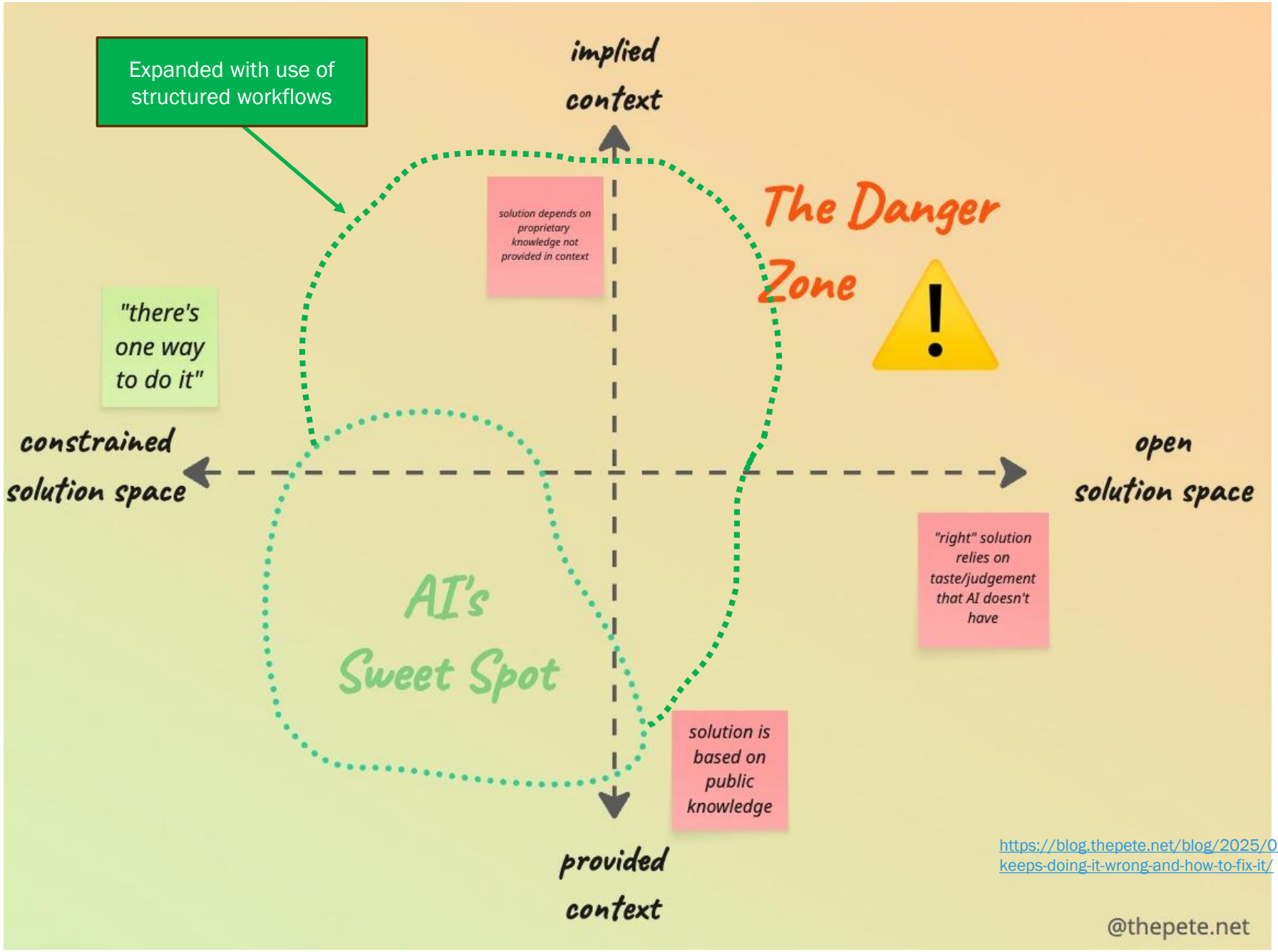
Building up one step at a time

Collaborating on current understanding and next steps

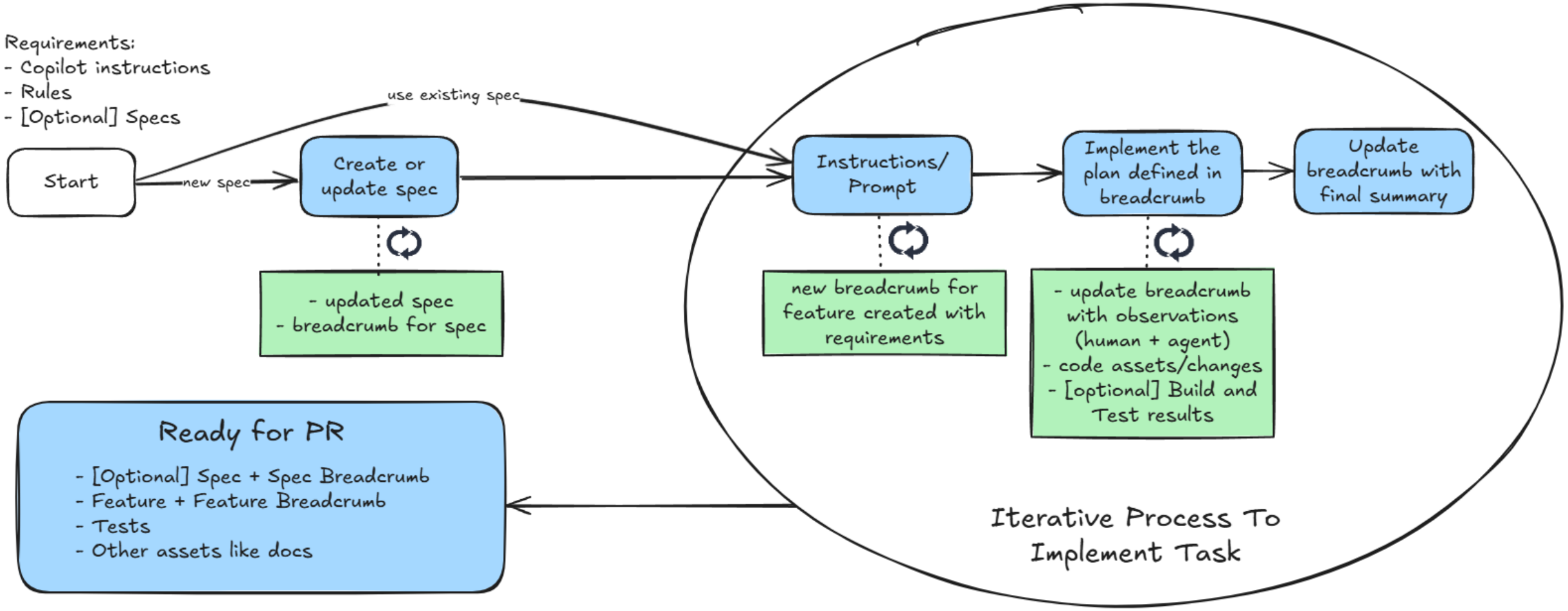
Works Alongside

SPECIFICATION DRIVEN WORKFLOWS

PLAN BASED WORKFLOWS



https://blog.thepete.net/blog/2025/05/22/why-your-ai-coding-assistant-keeps-doing-it-wrong-and-how-to-fix-it/


Requirements:
- Copilot instructions
- Rules
- [Optional] Specs
Start
use existing spec
new spec
Create or update spec
- updated spec
- breadcrumb for spec
Instructions/ Prompt
new breadcrumb for feature created with requirements
Implement the plan defined in breadcrumb
- update breadcrumb with observations (human + agent)
- code assets/changes
- [optional] Build and Test results
Update breadcrumb with final summary
Iterative Process To Implement Task
Ready for PR
- [Optional] Spec + Spec Breadcrumb
- Feature + Feature Breadcrumb
- Tests
- Other assets like docs

Breadcrumb Protocol

When working on tasks in this repository, please follow this collaborative workflow:

1. At the start of each new task, ask me for a breadcrumb file name if I haven't specified one.
2. Create the breadcrumb file in the `.copilot/breadcrumbs` folder using the format: `yyyy-mm-dd-{title}.md`
3. Use this breadcrumb file as our shared scratchpad for:
    - Requirements/asks from user [Required]
    - Planning approaches [Required]
    - Additional comments from user [Optional]
    - Documenting decisions [Required]
    - Sharing code snippets [Required]
    - Recording challenges and solutions [Optional]
    - Before and After Comparison [Required]
4. Update the breadcrumb file throughout our conversation as we make progress or change direction.
5. Refer back to existing breadcrumbs when relevant to maintain context across implementation sessions.
6. Update breadcrumb before start doing any other changes. Only proceed once the plan in breadcrumb is verified with user. Iterate this approach until task is done.

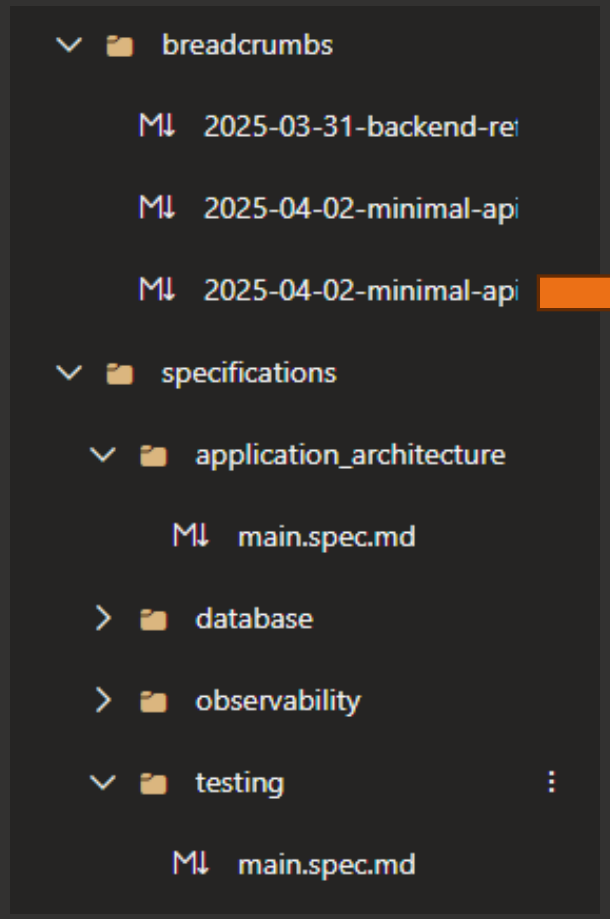
This practice creates a trail of decision points that document our thought process while building features in this brownfield repo, making future maintenance and onboarding easier.

Example usage: "Let's implement a new caching layer for the API responses" [Agent requests breadcrumb name] "Use 'api-response-caching'" [Agent creates `.copilot/breadcrumbs/2025-04-03-api-response-caching.md`]

https://github.com/dasiths/VibeCodingBreadcrumbDemo/blob/main/.github/copilot-instructions.md

General rules

- Use @terminal when answering questions about Git.
- Use specifications from the `./copilot/specifications` folder. Each folder under `specifications` groups similar specifications together. Always ask the user which specifications best apply for the problem if you're not sure.



2025-04-02-minimal-api-refactoring.md

Contents | Preview | History | Compare | Blame

Refactoring to Minimal APIs

Requirements/asks from user

- Refactor the existing backend code to use minimal APIs instead of controllers
- Follow the updated architecture specification in `.copilot/specifications/application_architecture/main.spec.md`
- Keep the same layered architecture structure but change the API implementation pattern

Planning Approach

- Review the current controller-based implementation
- Identify all controllers that need to be refactored
- Update Program.cs to use minimal API patterns
- Remove controllers and replace with endpoint registrations
- Test the refactored implementation
- Ensure all existing functionality is preserved

Current Analysis

The current implementation used traditional controllers:

- MakesController.cs
- ModelsController.cs
- VehiclesController.cs

These have been replaced with minimal API endpoint registrations in Program.cs.

Implementation Details

Program.cs Changes:

1. Removed `builder.Services.AddControllers()` since it's no longer needed
2. Removed `app.MapControllers()` since we're using direct endpoint mapping
3. Removed `app.UseAuthorization()` since it was causing errors and not required for our API
4. Added explicit minimal API endpoint registrations for all the endpoints:
   - GET /api/makes
   - GET /api/makes/{id}
   - GET /api/models
   - GET /api/models/{id}
   - GET /api/vehicles
   - GET /api/vehicles/{id}

Minimal API Implementation Examples:

After: Minimal API Pattern

```
// Makes endpoints directly in Program.cs
app.MapGet("/api/makes", async (GetAllMakesRequestProcessor processor) =>
{
    var response = await processor.HandleAsync(new GetAllMakesRequest());
    return Results.Ok(response.Results);
})
.WithName("GetAllMakes")
.WithOpenApi();

app.MapGet("/api/makes/{id}", async (int id, GetMakeRequestProcessor processor) =>
{
    var response = await processor.HandleAsync(new GetMakeRequest { MakeId = id });
    return response.Result is not null ? Results.Ok(response.Result) : Results.NotFound();
})
.WithName("GetMake")
.WithOpenApi();
```

Key differences:

1. **Code organization**: Controller code moved into Program.cs as direct endpoint registrations
2. **Boilerplate reduction**: No more need for controller classes, attributes, and action methods
3. **DI mechanism**: From constructor injection to direct parameter injection
4. **Routing**: From controller/action based routing to explicit path-based routing
5. **Response handling**: From ActionResult to direct Results types
6. **Metadata**: Using fluent API with `.WithName()` and `.WithOpenApi()` instead of attributes

Completion Status

- ✅ Analyzed the existing controller-based implementation
- ✅ Updated Program.cs to use minimal API endpoint registrations
- ✅ Removed controller files as they're no longer needed
- ✅ Successfully built and ran the application with the new minimal API patte
- ✅ Verified application starts up correctly and seeds data properly

DRAFT

Next Steps:

1. Add more comprehensive test coverage for the minimal API endpoints
2. Consider adding other HTTP methods (POST, PUT, DELETE) using the same pattern
3. Update client applications to ensure they work with the new API implementation
4. Consider further performance optimizations by leveraging features unique to minimal APIs

https://github.com/dasiths/VibeCodingBreadcrumbDemo/tree/main/.github/.copilot/breadcrumbs

Current Limitations

- Sometimes requires adding the .copilot folder as context.
- Tries to implement the next steps specified in breadcrumb occasionally even when not required.
- Not clear if agent picks up your changes in the next turn once the breadcrumb is updated manually. Requires pausing and asking the agent to look again.

Demo

https://github.com/dasiths/VibeCodingBreadcrumbDemo