CODE TRAINING

AI Assisted Software Development

Brownfield → Evergreen

John Michael Miller

CODE TRAINING

Senior Software Engineer at Code Staffing

Played roles of developer, architect, devops engineer, platform engineer, test architect, release manager

AI/ML Enthusiast and advocate for Effectively using AI to write code

- LinkedIn: `www.linkedin.com/in/johnmichaelmiller`
- Email: `john.miller@codestaffing.com`
- Blog: `https://codemag.com/blog/AIPractitioner`

Day Two & Three

CODE TRAINING

Working Effectively with Legacy Code, with AI assistance

- Going from Brownfield to Evergreen

But first...

https://githubuniverse.com

October 28-29

San Francisco, CA

In-person and virtual

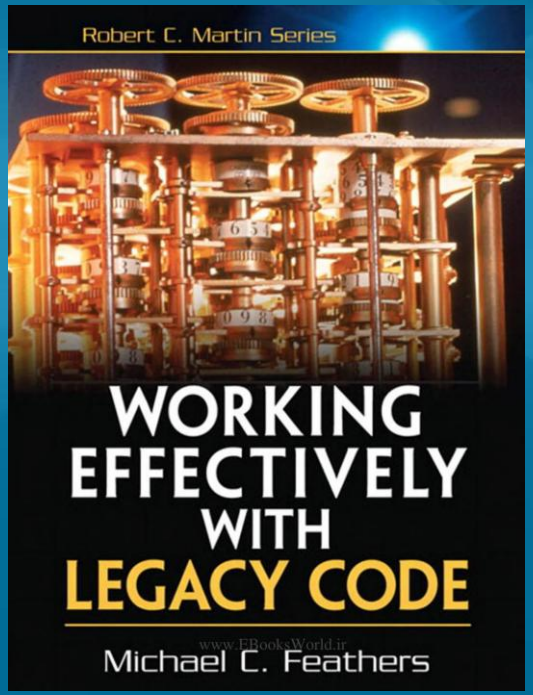
Agenda – Evergreen Development

CODE TRAINING

- Overview
  - What is and isn’t legacy code
  - What is evergreen code
- Before engaging Copilot
- Managing Copilot Context
  - Instructions, ChatModes, and Prompts
- Read Only Techniques
  - explain, review, document, test
- Modifying legacy code with AI assistance
- Managing Change Risk
  - small changes, refactoring, safety nets

Must Read

https://archive.org/details/working-effectively-with-legacy-code



Exercise #1.1 – Fork the repos

Objective: Fork the course repos

- Search GitHub for
  - AI-Assisted-Software-Development
- Fork the repo
  - This will create a personal copy under your GitHub account
  - You can make changes without affecting the original repo

Exercise #1.2 – Fork the target repo

CODE TRAINING

Objective: Fork the repo for the legacy code exercises

- Find a repo that could use some help
  - Choose a language you are familiar with
  - A personal project is cool, but a project you are not familiar with is better
  - Don’t use a company repo
  - Fork it
    - It allows you to experiment without affecting the original codebase

Prompt: I’m looking for a small GitHub Golang project that has a lot of open issues. Can you recommend one?

What is legacy code

CODE TRAINING

- No universally accepted definition
- Easier to define what is *not* legacy code

What is not legacy code

CODE TRAINING

- All codebases start as greenfield
- Few codebases are evergreen

Codebases degrade over time

Due to changes in:

- Technology
- Practices
- People
- Business rules
- Workflows
- Architecture

What is Evergreen Code

CODE TRAINING

- Evergreen code actively resists technical debt
- Evergreen ≠ Legacy; everything else is
- Evergreen is the goal
  - If we were to write all over again, it would turn out just like it is

Legacy code should instill respect not fear

CODE TRAINING

- It works - Don’t break it!
- It works - Well-tested (by users)
- It works - It is, what it is
  - “You get what you get and you don’t throw a fit”

Prelude – before engaging Copilot

CODE TRAINING

- Have a strategy for backing out of changes
  - Branching, Commit SHA, Archiving, Zipping
- Have a strategy for testing the code
  - No matter who or what wrote it
  - You need to have confidence that your test automation will catch issues before they reach production
  - If you don’t have that confidence, stop and build it first
- Have a strategy for reviewing changes
  - Peer review, AI review, Static analysis
  - You are going to be reviewing a lot of code. Make sure you have a plan to do it effectively
- Change sets as small as possible
  - Make incremental changes and validate frequently
  - AI can get carried away. Don’t let it.

Managing Copilot

- Treat Copilot as a highly skilled, junior developer who needs effective management to succeed
- Clear instructions + careful review = great outcomes

Managing Context

- The importance of context
  - Too little
  - Too much
- Prompts
  - Crafting effective prompts
  - Prompt engineering
- Adding context
  - Instructions markdown
  - File-based custom instructions, automatic context injection, instruction scopes, structure and authoring best practices, structure and authoring best practices
- Custom chat modes
  - Mode definition, tool integration, quick mode switching, workspace and user scope, persona-driven workflows, enhanced productivity and consistency

Awareness

- Copilot has a limited context window (token budget)
- Long prompts/files can push useful info out of scope
- Premium usage can be tracked monthly (dashboard)

Prompt Design

CODE TRAINING

- Keep prompts short and modular
- Use step-by-step requests
- Reuse prompt files/templates
- Reset chat if answers drift

Context Management

CODE TRAINING

- Use #-mentions to pull in files, folders, or symbols
- Leverage Spaces or Knowledge Bases for docs
- Close irrelevant tabs to reduce noise

Monitoring

- Premium Request Dashboard → usage reports
- Token Estimation Tools (e.g., OpenAI Tokenizer)
- Watch for off-topic replies = context overflow

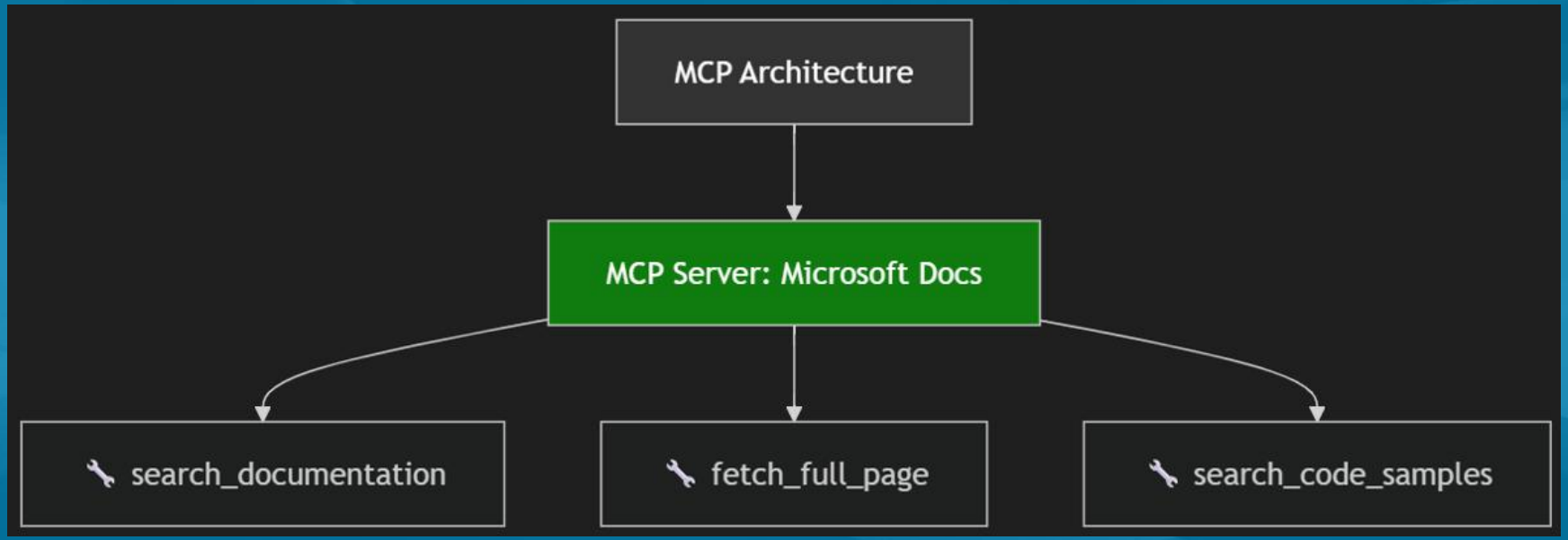
Coming Soon

- Real-time token counters (requested)
- Context usage indicators (progress bars, % left)

Key Takeaways

- **Claude Sonnet 4.5** → best for ultra-long context (200k+)
- **GPT-5** → strong reasoning + 128k input / 16k output
- **Copilot Chat (GPT-4o)** → 64k tokens, tuned for IDE workflows
- Model choice = context size + reasoning style trade-off

MCP Servers and Tools



MCP Architecture

Adding Context - Tools

CODE TRAINING

- Configure Tools...
- Built-in and not
- Selecting and Unselecting Tools
  - 128 Tool limit

Raising the Bar – with Instruction Files

CODE TRAINING

- What are instruction files?
- Provide context and guidance to Copilot and
- Can include:
  - Project-specific instructions
  - Tech-specific instructions
  - Standards and practices
- They need to be consistent and not contradictory

Instruction File Locations

- Personal - `$home\.github\copilot-instructions.md`
- Project - .github
- File names should be `*.instructions.md`

Meta Instructions

CODE TRAINING

Instructions File	Description
ai-assisted-output.instructions.md	Instructions for generating AI-assisted output
copilot-instructions.md	Instructions for using Copilot effectively
instruction-prompt.instructions.md	Instructions for creating effective prompts
prompt-file.instructions.md	Instructions for prompt files that generate instructions

Copy these files into the .github/instructions folder of your project to guide Copilot in the creation of prompts that create instruction files.

AI-Assisted Output Policy — At a Glance

- Purpose: enforce provenance for all AI-assisted artifacts
- Requires: embedded YAML front matter (ai_generated, model, operator, chat_id, ai_log, timestamps)
- Workflow: new chat → `ai-logs/yyyy/mm/dd/<chat-id>/conversation.md` + `summary.md`
- Enforcement: CI/PR checks should fail files with `ai_generated: true` missing `chat_id` or `ai_log`

See: .github/instructions/ai-assisted-output.instructions.md

GitHub Copilot Instructions — At a Glance

CODE TRAINING

- Scope: Guidance for GitHub Copilot usage and AI provenance across the repo
- Model rule: always use underlying model format “provider/model@version”
- Metadata: embed YAML front matter (ai_generated, model, operator, chat_id, prompt, started, ended, task_durations, total_duration, ai_log, source)
- Chat/logging: New chat → new ai-logs/yyyy/mm/dd//conversation.md + summary.md
- Post-creation: update ai-logs, create summary.md, and add README entry for notable artifacts
- CI check: fail PRs when ai_generated artifacts lack chat_id or ai_log or referenced ai_log missing

See: .github/instructions/copilot-instructions.md

Generating the Prompt for Evergreen Software Development

Meta Prompt: Create a prompt file that generates an instruction file for evergreen software development

- Keep `main` buildable, secure, releasable
- Automate dependency updates & CI (Dependabot, SCA)
- Merge small, frequent changes; pin and monitor deps
- PR checklist: CI green, security review, changelog
- Enforcement: GitHub Action “Evergreen check” (deps + tests)

Generating the Evergreen Instruction Files

Prompt: #file:create-evergreen-instructions.prompt.md create the instruction file for evergreen software development

- Prompts to generate instructions
  - Instructions on generating instruction files
- Use current codebase as context
- Keeping instruction files fresh
- Prompts for updates
- Copilot suggestions based on codebase changes

Meta Prompts

CODE TRAINING

Exercise #3 – Create Instruction Files

Objective: Create instruction files manually and with Copilot assistance

- Manually create instructions with:
  - Business rules
  - Workflows
  - Purpose
  - Tech stack
  - Architecture
- Use Copilot to generate instruction files using the codebase
- **Bonus**
- Review for errors, omissions and inconsistencies
- Ask Copilot for updates instructions based on tech/practice changes

Day Three – Legacy Development

CODE TRAINING

But first...

Installing the Azure DevOps MCP Gallery Extension

- Enable MCP Gallery in VS Code Settings
  vscode://settings/chat.mcp.gallery.enabled
- Search extensions for
  - @mcp azure devops

Understanding legacy code

CODE TRAINING

- Generating documentation
- Update README files
- Generate architecture diagrams
- Identify and explain complex or confusing code

Exercise #4 – Understanding legacy code

CODE TRAINING

Objective: Create documentation for a legacy project using Copilot.

- Update or create:
  - Development guide
  - Deployment guide
  - README
  - Architecture diagrams
- Execute prompts to update documentation
- Review for errors and omissions
- **Bonus**: Use another model to review the documentation

Meta-Instructions

CODE TRAINING

Keeping Code Evergreen

CODE TRAINING

Prompt:

- Create an instruction file for an evergreen project. put it in the .githubfolder. It should be technology agnostic
- Create an instruction file for an evergreen golang backend. put it in the .githubfolder. It should be technology focus on golang and not repeat instructions in the #file:evergreen.instructions.md
- Review the code in the Fundamentals.sav.calculator folder and create issues for bringing the in compliance with the instructions. Prioritize and categorize the issues.

Creating a safety net

CODE TRAINING

- Linting
- What AI can do beyond static tools
- Create and execute linting rules with AI

Test automation

CODE TRAINING

- Unit, integration, E2E, and other tests
- Generating tests
- How many tests are enough?

Exercise #5 – Generate Tests

Objective: Same as earlier—focus on test creation and review.

Bonus: Use another model to review the tests

Identifying Instruction Compliance Gaps

- Compare current implementation to instructions
- Generate issues from gaps
- Identify low-hanging fruit

Exercise #6 – Identify Gaps

Objective: Use Copilot to compare the implementation with the instruction files.

- Create prompt to identify deviations
- Review response

Bonus: Generate GitHub issues from findings

Exercise #7 – Address a Compliance Gap

Objective: Create a plan to address a deviation found in Exercise #6.

- Ask Copilot to create a remediation plan
- Review the plan

Bonus: Ask a different model to review the plan

Comparing Implementations

CODE TRAINING

- Implement the same change with multiple models
- Ask Copilot to summarize pros and cons

Exercise #8 – Compare Implementations

CODE TRAINING

Objective: Use two models to implement a change and compare results.

- Create branches for each model
- Implement fix using different models
- Compare implementations

Bonus: Ask a second model to compare and report pros/cons

Creating Instructions

CODE TRAINING

- What are instructions?
  - Locations:
  - Project
  - User
- AI-assisted instruction generation

Wrap-Up and Open Q&A

CODE TRAINING

- Review the day’s learning outcomes.
- Address questions or concerns.
- Discuss next steps and future learning opportunities.