Global Shop Solutions Software Modernization AI Guide

***How to Use Cursor, ChatGPT, and Codex in Our Day-to-Day Development***

**Our Environment and Goals**

**We build and maintain COBOL programs using:**

* **Visual Studio 2015 with Fujitsu licenses**
* **Tortoise SVN for version control**
* **Internal Clinics for testing and running programs**

We’re not replacing what already works.  
Our goal is to **use AI tools to speed up our work, improve code clarity, and help with documentation**, all while staying within our existing workflow.

**Using Codex, Cursor, and ChatGPT**

| **Tool** | **What It Is** | **Best Use** |
| --- | --- | --- |
| **Codex** | Developer-focused model optimized for code reasoning, refactoring, and automation.  (Accessed through CLI / Web) | Deep code understanding across files, tests, and architecture. |
| **Cursor** | AI-powered IDE (VS Code-based) with native AI editing and inline commands | Fast, contextual editing and refactoring within your codebase |
| **ChatGPT Enterprise** | Full-featured conversational environment with advanced reasoning, larger context, and document upload | Research, planning, documentation, architectural ideation, and integration with Company Documentation (Sharepoint connector) |

**Think of it this way:**

* **Cursor** = your pair programmer
* **ChatGPT** = your senior architect / research assistant
* **Codex** = your junior dev team handling delegated work

**Quick Reference Cheat Sheet**

* **Project Setup:**
  + *Rules:* Define in .cursor/rules/ to enforce guidelines (e.g. coding style, terminology). Always rules apply to every query.
  + *Memory Bank:* Initialize with npx cursor-bank init (optional) to get structured context file. Update these files as you work (update memory bank command) to keep AI’s knowledge current.
  + *Model Settings:* Use GPT-4 or Claude Sonnet 4.5 for complex tasks. Enable “Thinking” mode for long reasoning tasks if needed.
  + *Indexing:* Cursor auto-indexes files for context. Use **Resync Index** after large changes.
* **Prompting Tips:**
  + Be explicit about **what you want** and **provide context** (e.g. attach code with @Files or snippets) [cursorpractice.com](https://cursorpractice.com/en/cursor-tutorials/getting-started/6-Context#:~:text=%40Files).
  + Use **Plan/Act** workflow: "Plan: <task>" -> get plan, then "Act" to execute.
  + **Refer to rules/memory** in prompts (e.g., “following our coding standards in systemPatterns.md…”).
  + Ask for step-by-step reasoning if needed (the AI can “think aloud”).
  + If output is off, **clarify or rephrase** and try again. Small tweaks in wording can yield better results.
* **Cursor Shortcuts & Features:**
  + @Files, @Code, @Folders: Include code or entire files in prompt.
  + @Docs: Include documentation (after indexing it via Settings > Docs).
  + Drag & drop files or use **Ctrl+Shift+L** to add selected code to chat.
  + **Ctrl+Shift+P** then “Reload Window” if Cursor misbehaves or AI seems confused.
  + Keep .cursorrules open in a tab to help AI keep context, especially during reloads.
  + Use **Add Context** window to pin memory files if needed (ensuring they’re always sent).
* **Codex CLI:**
  + Install via npm install -g @openai/codex. Run codex and log in with ChatGPT Enterprise.
  + In Codex CLI, type natural language commands. Use /approvals read-only to plan or /approvals auto for default mode.
  + Use /model gpt-5-codex for the advanced code model if available.
  + Press Ctrl+C to interrupt if it’s going astray; you can then refine your instruction.
  + Codex CLI can run shell commands: e.g., “Compile and run program X” – it will attempt to do so. Always review its actions (it will ask for confirmation for safety).
* **ChatGPT Enterprise:**
  + Data is private (not used for training by default), so you can safely discuss internal code.
  + Use it for Q&A, general guidance, or even to generate **regexes, scripts, SQL queries** etc., that might assist around the code (e.g., an SQL to extract test data).
  + Enable code interpreter (if available) for data tasks – though not directly needed for COBOL, it’s great for analyzing logs or converting data formats as a side tool.
* **Troubleshooting:**
  + *AI Hallucinating or off-track?* – Refresh context: open relevant files, reload Cursor. Verify that memory files contain correct info (no “poisoned” old info).
  + *Cursor AI unresponsive?* – Possibly re-login or ensure API keys haven’t expired. Try a smaller prompt to see if it responds. If needed, restart Cursor IDE.
  + *Context lost mid-conversation?* – Summarize where you left off, start a new chat, feed summary back in.
  + *Large output gets cut off?* – The model hit token limit for response. You can prompt “continue from where you left off” and it often will. For code, ensure it didn’t omit any part.
  + *Getting rate-limited or errors from API?* – Space out requests; enterprise should have high limits, but if doing heavy usage (like Codex running long sessions), you might hit caps. If so, coordinate usage or contact admin about quota.

**Cursor**

**Rules (**https://cursor.com/docs/context/rules)

* Rules provide system-level instructions to Agent and Inline Edit. They are persistent context, preferences, or workflows for your projects. It’s good practice to have rules files for all codebases. These files live in ./cursor/rules folder in your root project directory.

For more versatility when using other AI tools/agents and want them to adhere to the ‘rules’ aspect, you can utilize AGENTS.md  
AGENTS.md is a simple markdown file for defining agent instructions. Place it in your project root as an alternative to ./cursor/rules for straightforward use cases.

* Unlike Project Rules, AGENTS.md is a plain markdown file without metadata or complex configurations. It's perfect for projects that need simple, readable instructions without the overhead of structured rules.
* Cursor supports AGENTS.md in the project root and subdirectories. So each level can be specifically tailored to that section of your project

**A screenshot of a computer

AI-generated content may be incorrect.Giving Cursor access to codebase**

Adding directories to your workspace so you Cursor can have access and reference other programs outside of you WIP that might provide better context to your project (i.e. copybooks, calling programs, etc).

Just right-click and Add Folder to Workspace. And you can now   
  
reference what is in trunk when you chat with cursor

**“Memory (for the next agent)” and “Change History”**

This idea is to keep a running “brain” (Memory) so the next person/agent starts with context, and a “black box recorder” (Change History) so we can audit exactly what changed and why.

**Set up the two docs in your workspace root:**

* PROJECT\_MEMORY.md (for durable context the next agent needs)
* CHANGE\_HISTORY.md (append-only timeline of proposed and approved changes)

**Starter templates:**

|  |  |
| --- | --- |
| **CHANGE\_HISTORY.md**  # Change History (Append-Only)  ## YYYY-MM-DD HH:MM (Author/Agent)  ### Intent  - One precise change (scope & motivation)  ### Proposed Diff (before -> after)  ```diff  - old line(s)  + new line(s) | **PROJECT\_MEMORY.md**  # Project Memory (Durable Context)  ## Problem Space  - What we’re trying to solve:  - Business rules:  ## Architecture / Data Flow (short)  - Systems touched:  - Key files/modules:  - Inputs/outputs:  ## Known Patterns & Gotchas  - Reusable routines:  - Known pitfalls:  ## Glossary  - Term -> meaning  ## Open Questions  - [ ] Question -> owner |

For Cursor to keep reference to these 2 files, it would be a good idea to update your rules or AGENTS.md file  
  
Example of AGENTS.md file:  
## Agent Behavior: Synchronizing with Change History & Project Memory

### 1. Monitor existing docs

- Before starting new work, the agent \*\*must read\*\* `CHANGE\_HISTORY.md` and `PROJECT\_MEMORY.md`.

- Use them as \*\*reference\*\* to avoid repeating previous ideas or conflicting with past decisions.

### 2. Use history & memory in reasoning

- When proposing changes, check if a similar change was already made (per `CHANGE\_HISTORY.md`).

- Use `PROJECT\_MEMORY.md` to recall project conventions, patterns, naming, and constraints.

- If new insight is discovered (edge case, rule, pattern), propose an update to memory.

### 3. How to update docs

- \*\*CHANGE\_HISTORY.md\*\*: For every proposed or accepted change, append a new entry.

- Include date, your identity, intent, diff, impact, rollback plan, and status.

- \*\*PROJECT\_MEMORY.md\*\*: For stable, reusable knowledge (not one-off decisions), propose adding or refining entries.

- Do \*not\* erase or override past memory entries without clear rationale.

### 4. Guardrails & integrity

- Never delete or rewrite prior entries in `CHANGE\_HISTORY.md` or older memory.

- All updates should be \*\*additive or corrective via append\*\* (unless explicitly asked).

- Always reference file paths, versions, and line ranges when writing history or memory entries.

### 5. Workflow placement

- At session start: agent reads both docs before accepting further prompts.

- Before proposing changes: agent cross-checks history & memory.

- After producing a diff: agent outputs the new entry blocks for both docs.

- In multi-turn tasks: agent preserves memory-awareness across turns.

**Targeting files, folders, tabs, and selected code (context precision)**

You can target specific files in your Chat Window with the agent by:  
@ - Using @[fileName] will target that entire file as something to reference

# - A specific tab you have open that might not be in your current workspace

Highlight Code – Highlighting code chunks and then clicking ‘Add to Chat’

**Version Controlling Cursor (Or any other AI Agents)**

When working with AI, which can sometimes go very rogue and hallucinate bugs into code if the context window gets overflown, it’s a good idea to version control every change that works.

In our case, it might be best to have a separate WIP away from our Checkout repo that we can setup with our own private SVN session to ensure we incrementally moving in the right direction. Also to revert when AI goes haywire.

**Spec, Plan, Code, Review**

Another good practice to try is to Spec out what it is your coding and document that into a file for the agent to reference. This helps keep the AI focused on exactly what it should be working on or looking at.  
Then have AI specifically plan out the next move before executing anything. Tell it to thoroughly generate a gameplan and build a ToDo list that it will adhere to.

If that looks good to you, then execute coding.

A good idea is to open a new chat instance and have that AI ‘Code Review’ the changes the previous agent just made and adjust accordingly. This helps from the previous AI trying to ‘gaslight’ you or convince you why it’s right.

**ChatGPT**

**Personalization:**

Custom Instructions, a feature in the Settings menu where users provide information about themselves and specify desired response styles, tones, and formats. This allows ChatGPT to act as a more tailored and effective assistant by reflecting the user's needs, work style, and preferences without needing to repeat them in every prompt.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

Example:  
Act as my expert assistant with access to all your reasoning and knowledge.

Always provide:

A clear, direct answer to my request.

A step-by-step explanation of how you got there.

Alternative perspectives or solutions I might not have thought of.

A practical summary or action plan I can apply immediately.

Never give vague answers. If the question is broad, break it into parts. If I ask for help, act like a professional in that domain (teacher, coach, engineer, doctor, etc.). Push your reasoning to 100% of your capacity.

Ask me questions until you’re 95% confident you understand what I’m looking for.

A screenshot of a computer program

AI-generated content may be incorrect.**Projects:**

Projects are dedicated "smart workspaces" or organizational folders for long-running tasks, grouping relevant chats, uploaded files, and custom instructions to maintain context and streamline workflows. It allows you to add reference documents, and organize existing conversations to help the AI stay on-topic and provide more relevant responses. 

**Codex**

CLI Codex setup

1. Download NodeJS if you do not have it already:  
   <https://nodejs.org/en/download>
2. Open a terminal or powershell and type:  
   npm install -g @openai/codex
3. Navigate to directory you want to work in and type:  
   codex
4. Authenticate your GSS ChatGPT account

Web

* <https://chatgpt.com/codex>

OR

* Accessing it through ChatGPT itself

A screenshot of a computer

AI-generated content may be incorrect.

Cursor (Extension Install)

A screenshot of a computer

AI-generated content may be incorrect.