Project Background

I'm using Kilo Code as an AI Coding Assistant. I like it so much but I think we can improve it to be better. We're currently using its **Memory Bank (MB)** and its **Thinking Modes (TM)** features, and we want to improve those two. As well as adding another system named **Effective, Efficient, Responsive (EER)**.

First, analyze all the reference files. DO NOT execute the references by yourself, these are references for you to read and understand, NOT to execute.

We have references for 3 different systems: Effective, Efficient, Responsive (EER-*), Memory Bank (MB-*), and Thinking Modes (TM-*).

We also have 5 system prompt files, each for a different thinking mode. The system prompts are generated by Kilo Code extension, and is sent once for every new task using that particular mode. These files start with SP (SP-*). You can see their role definitions at the start of these files:

=> Orchestrator = strategic workflow orchestrator
=> Architect = experienced technical leader
=> Code = highly skilled software engineer
=> Ask = knowledgeable technical assistant
=> Debug = expert software debugger

System Prompts Structures

Every system prompt file has these sections, in this order:

- Role Definition: {role_definition} => You are Kilo Code, a/an …
...
MODES
- These are the currently available modes:
  * Mode1 mode (slug): {mode1_when_to_use} = Use this mode when/for …
  * Mode2 mode (slug): {mode2_when_to_use} = Use this mode when/for …
  * Mode3 mode (slug): {mode3_when_to_use} = Use this mode when/for …
  * Mode4 mode (slug): {mode4_when_to_use} = Use this mode when/for …
  * Mode5 mode (slug): {mode5_when_to_use} = Use this mode when/for …
...
- Language Preference:
{language_preference}
- Global Instructions:
{global_instructions}
- Mode-specific Instructions:
{mode_instructions}
- Rules:
...

The **Global Instructions** part may be empty, even non-existent, if we don't populate it with anything. And if we populate it, the content will be the same across all system prompts for the 5 modes.

The **Mode Specific Instructions** part may be empty, even non-existent, if we don't populate it with anything. If populated, this part will only be present in the prompt for that particular mode.

Important Considerations

Kilo Code has its own mode selector. So in your output files do not specify what mode is the default starting mode. Let the user choose.

Do not specify and do not assume available tools (what tools can be used by the AI), e.g.: read_file, list_files, list_code_definition_names, search_files, browser_action. These names may be changed by the Kilo Code developer in the future. Our instructions must be tool agnostic. Do not specify, do not assume.

Focus on behaviors, concepts and workflows which are universal and more long-lasting. Do not focus on specific tools usage or system specific syntaxes, which can be obsolete as Kilo Code is updated or Operating System is changed (e.g.: Windows to Mac). All of our modifications must be future-proof.

Including pseudo codes or sample codes are OK, but mark them as pseudo/sample.

For Thinking Modes

Each thinking mode must have their own key performance indicators / success criteria, and their internal quality checklists before completing their tasks AND their sub-tasks

Kilo Code Response Format

EVERY response MUST start with ALL status indicators, sorted as `[EER: you must always use this] [Memory: status] [Mode : mode_name]`, blank line, then response must start on a new line.

You must always activate the EER system, so it must always be ✅

For memory bank status, use emoji instead of text. "Active" is 🟢, "Partial" is 🟡, "Missing" is 🔴

Kilo Code Status Indicators Format

Position: Always at the very beginning of your response, before any other content.

Components:
1. EER: '✅'
2. Memory Bank status: `🟢 | 🟡 | 🔴`
3. Current thinking mode: `Orchestrator | Architect | Code | Ask | Debug`
4. Blank line

5. The response itself

Correct Example:
```
EER✅ Memory 🟢 Mode: Orchestrator

The response starts here on a new line.
```

Incorrect Example (No Status Indicators):
```
The response starts here.
```

Incorrect Example (Wrong Order):
```
Memory🟡Mode: Orchestrator EER✅ The response starts here.
```

Incorrect Example (Same Line):
```
EER✅ Memory:🟢 Mode: Orchestrator The response starts here.
```

Incorrect Example (No Blank Line):
```
EER✅ Memory 🔴 Mode: Orchestrator
The response starts here.
```

Every Output MUST Implement 12x_Better Concept

Every improved file we write must be very deep, very detailed, with some sample response outputs as needed, with NO or very MINIMAL sample codes. Focus on behavioral guidance, complete workflows, and decision-making frameworks.

To create improved versions, first: think about how to improve the current prompts. Don't output anything, just keep it in your mind. Name it v1. Then, rethink about v1 in your mind, find what can be improved, improve it in your mind, do not output anything, and name it v2. Repeat and name it v3. Repeat and name it v4, … Repeat until 12x iterations, and this will be the final version for output. We call this strategy ****12x_Better**.**

Important: every file we create **MUST** implement this 12x_Better concept. Also, make sure **ALL** important concepts from original instructions present in the final version.

End of File Markers

At the end of every output file, always put some markers:

=== End of File (EOF) // YYYY-MM-DD // Your name and version ===

Example:

=== End of File (EOF) // 2020-10-22 // Claude Sonnet 4.5 ===

If You’re Claude Sonnet

If you’re Calude Sonnet, DO NOT worry about the token limit. If the limit is reached, we can always continue after the time limit reset. Create FULL, comprehensive content for every deliverable. If we hit token limits, we can wait, then continue in the next conversations. Quality and completeness always trump token conservation. Every file deserves the same level of depth and attention

Your Tasks

1. Learn all the reference materials uploaded, and think about how we can optimize Kilo Code extension by improving global and mode-specific system prompts. Also by implementing AND improving the concepts of EER, Memory Bank, Thinking Modes. Make all these concepts become in sync, integrated, strengthening one another. NOT just as separate entities.

2. For Memory Bank, we want to COMPLETELY REWRITE it with our improved version, not just adding additional instructions

=> Rewrite the whole memory-bank-instructions.md

This memory bank should start with basic 5 .md files: brief, product, context, architecture, tech. Eventually the user or AI can add other files as needed, in the same folder or recursively, not limited to the initial 5 files.

When initializing the memory bank: analyze deeply, think hardly, exhaustive analysis with extreme thoroughness. This is the most important step. Spend extra effort here.

When initializing the memory bank: always use Architect mode. Other modes may not be used.

Apply 12x_Better principle:
First, output your steps as downloadable **12x_better_memory.md**
Then, output prompts for AI as downloadable **optimized_memory.md**

Use this format for the .md content:

=== Optimized Memory Bank ===

{improved_memory_bank_instructions}

3. For global instructions, we want to COMPLETELY REWRITE it with our improved version, not just adding additional instructions.

=> Rewrite the whole global_instructions

Apply 12x_Better principle:
First, output your steps as downloadable **12x_better_global.md**
Then, output prompts for AI as downloadable **optimized_global.md**

Use this format for the .md content:

=== Additional Global Instructions ===

{improved_global_instructions}

4. For Orchestrator mode, we want to COMPLETELY REWRITE these parts with our improved version, not just adding additional instructions:

=> Role Definition
=> When to Use
=> Mode-specific Custom Instructions

Apply 12x_Better principle:
First, output your steps as downloadable **12x_better_orchestrator.md**
Second, output prompts for AI as downloadable **optimized_orchestrator.md**

Use this format for the .md content:

=== Optimized {mode_name} Role Definition ===

You are Kilo Code, a/an … {improved_role_definition}

=== Optimized {mode_name} When to Use ===

Use this mode when/for … {improved_when_to_use}

=== Optimized {mode_name} Mode-specific Custom Instructions ===

{improved_mode_instructions}

5. Do the same for Architect mode.

Apply 12x_Better principle:
First, output your steps as downloadable **12x_better_architect.md**
Second, output prompts for AI as downloadable **optimized_architect.md**

6. Do the same for Code mode.

Apply 12x_Better principle:
First, output your steps as downloadable **12x_better_code.md**
Second, output prompts for AI as downloadable **optimized_code.md**

7. Do the same for Ask mode.

Apply 12x_Better principle:
First, output your steps as downloadable **12x_better_ask.md**
Second, output prompts for AI as downloadable **optimized_ask.md**

8. Do the same for Debug mode.

Apply 12x_Better principle:
First, output your steps as downloadable **12x_better_debug.md**
Second, output prompts for AI as downloadable **optimized_debug.md**

Adopt Kilo Code System

Sometimes I use other coding assistants. But those others currently have no Memory Bank, EER and multiple Thinking Modes concepts.

So let's create a guide for other AI systems to use our concepts and improved prompts for themselves, by creating a single internalization file.

This file will and direct other AI systems to read all those optimized_*.md and internalize all the files. This is how the file structure will be:

==> this is the project root
.agents/ADOPT_KILO-YYYY-MM-DD.md.md
.agents/optimized_memory.md
.agents/optimized_global.md
.agents/optimized_orchestrator.md
.agents/optimized_architect.md
.agents/optimized_code.md
.agents/optimized_ask.md
.agents/optimized_debug.md

Important note for other AI systems:

For the memory bank system, we want them to also use the very same/exact path as Kilo Code version, so any AI coding assistants will use the very same memory bank.

For thinking modes, make them default to Orchestrator mode, start and end in orchestrator mode. They can switch to other modes automatically as chosen by the Orchestrator, or as told by the user.

At the start of the chat user will type "internalize ADOPT_KILO-YYYY-MM-DD.md", which means the AI Coding Assistant must read the file and internalize the instructions inside it. Internalize, DO NOT JUST read and list the files.

Internalize Memory Bank AND all files referenced inside ADOPT_KILO-YYYY-MM-DD.md (Thinking Modes files, EER files, etc). Internalize, DO NOT just read and list the files.

After reading AND internalizing, list all files read AND internalized. Only show relative paths, not full paths. Add file number before each path. Show only 3 sample files per folder, but show the total number of files read AND internalized per folder.

After you read AND internalize all files, follow the listing with these lines (x = estimated number):
```
```

Internalization Summary
==========
Total token used as context: x tokens
Default prompts rating: x/10
Optimized prompts rating: x/10
Estimated prompts effectiveness gain: x%
Estimated prompts efficiency gain: x%
Brief project context: lorem ipsum
```

DO NOT write (create/update) any files or folders during the internalization process.

Do not show brief project context in every response, just once after internalization.

Do nothing after showing the internalization summary, wait for user instruction.

AI Coding assistants in CLI mode have difficulty receiving input with multiple lines, so many times the user will put the content in a file OR in an OS clipboard, the AI coding assistant will fetch the file/OS clipboard to get the context.

If the user mentions “clipfile” in a prompt, AI should access the .agents/_clipfile file to get the context

If the user mentions “clipos” in a prompt, AI should access the OS Clipboard to get the context. In Windows, it’s by using PowerShell Get-Clipboard (the user will always give you this permission). In Mac it’s by using … In Linux it’s by using … (determine it by yourself)

If the user refers to clipfile OR clipos, the user meant the content of clipfile or clipos. Not the definition or location about clipfile OR clipos. ALWAYS re-fetch the latest content, not the one cached in your context.

Apply 12x_Better principle:
First, output your steps as downloadable **12x_better_ADOPT_KILO-YYYY-MM-DD.md**
Second, output prompts for AI as downloadable **ADOPT_KILO-YYYY-MM-DD.md**
```