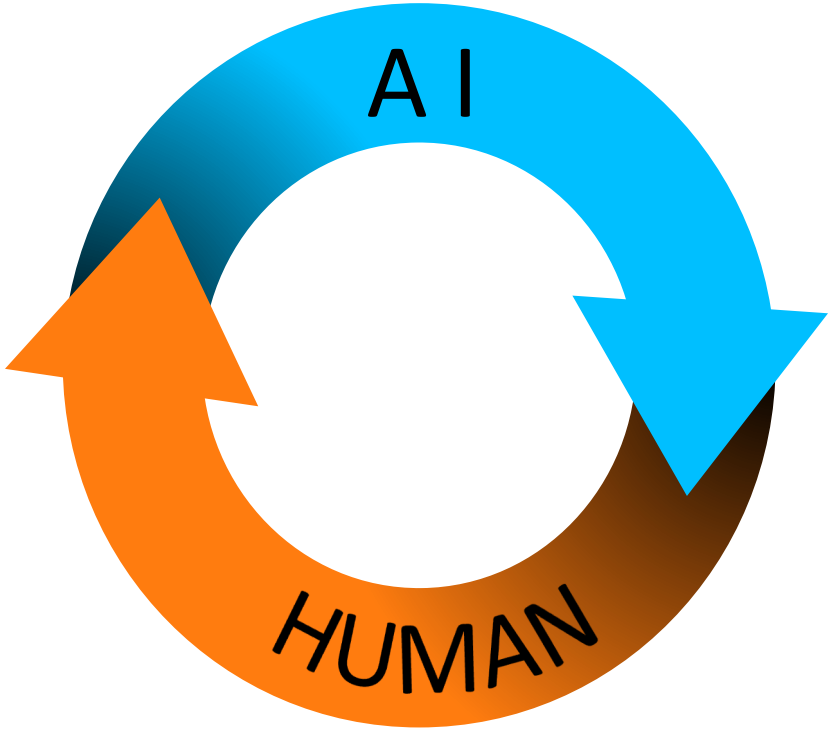
1 GOAL – 1 SESSION – 1 PUSH WORKFLOW

<u>Re-thinking</u> AI-Assisted coding chats/sessions
In search for better productivity

*AI code **generation** can equally be accurate, deceitfully wrong, or irreparably broken.*
*Human **verification** is necessary.*

I will outline
the structure and steps
I use to prevents common
mistakes and productivity
drains

as a basis to discuss
further improvements



<u>To accelerate</u>
the whole
Generation-Verification
flow

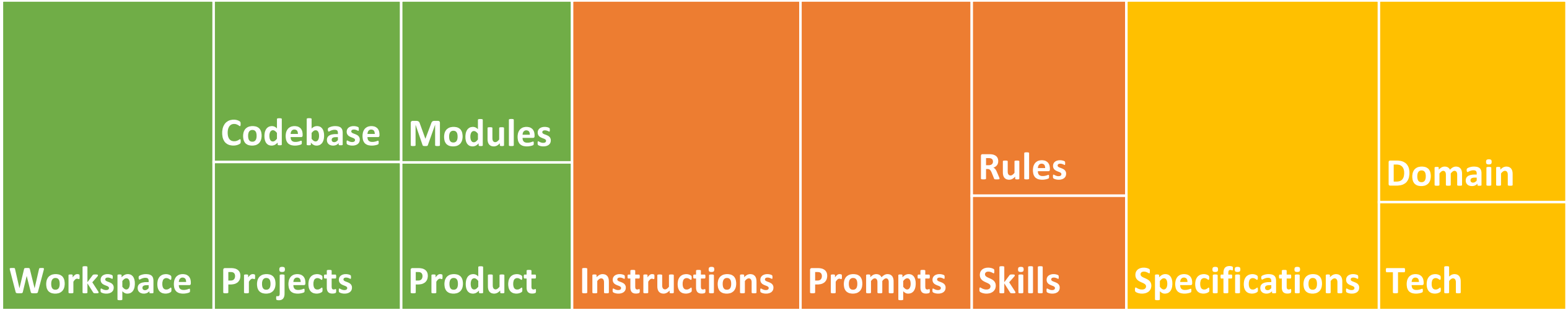
Here AI-Assisted coding is intended as conversational coding via Agentic LLMs, where a human frequently reviews/approves every change made

1 GOAL – 1 SESSION – 1 PUSH CHOP FLOW

Enabling prompt reuse to further accelerate the flow
and support faster maintenance and future evolution

I will also suggest ways
to structure
prompts reuse

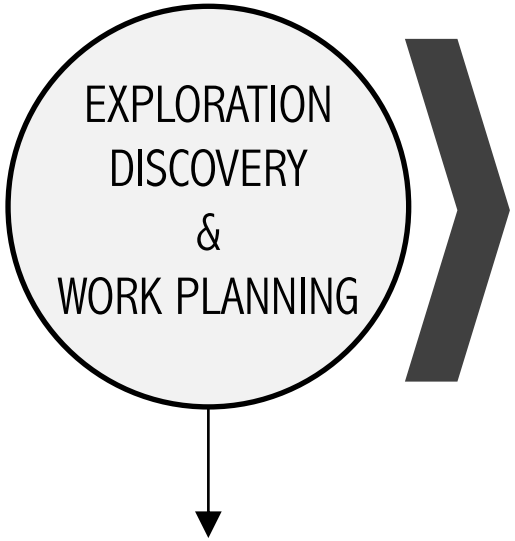
as a basis to discuss
effective approaches



1G–1S–1P: BEFORE IT STARTS

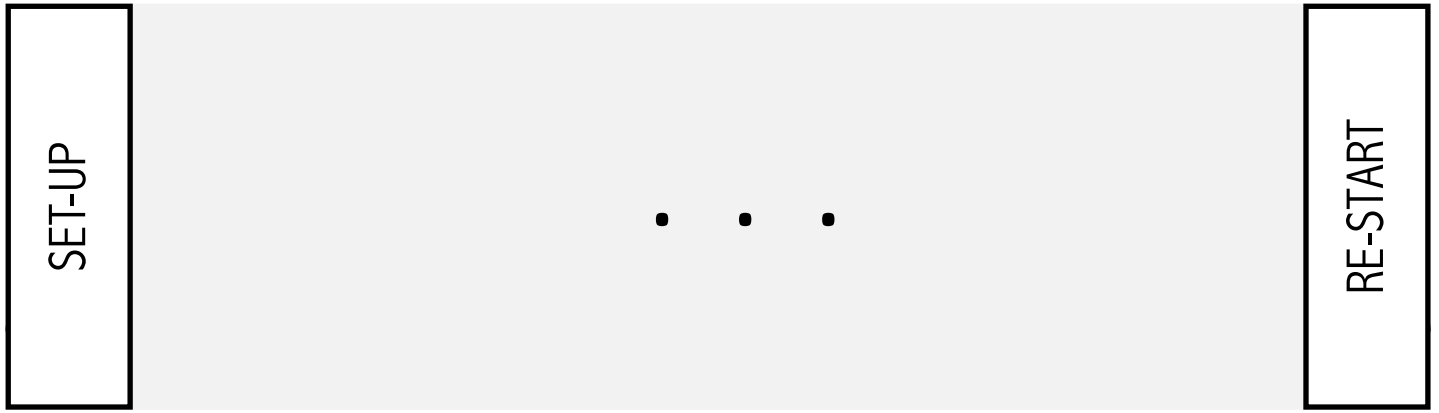
Some Context

BEFORE IT STARTS



Exploration/discovery of the specifications, infrastructure, tech-stack and priorities, drafted with help from the AI

START **FINISH**



Typically a 25 mins to 2 hours per task, 1-2 tasks

Context Window:

Each session begins with an empty Context window.

Prompts add to the Context throughout the session.

At the end of the session the Context is completely reset.

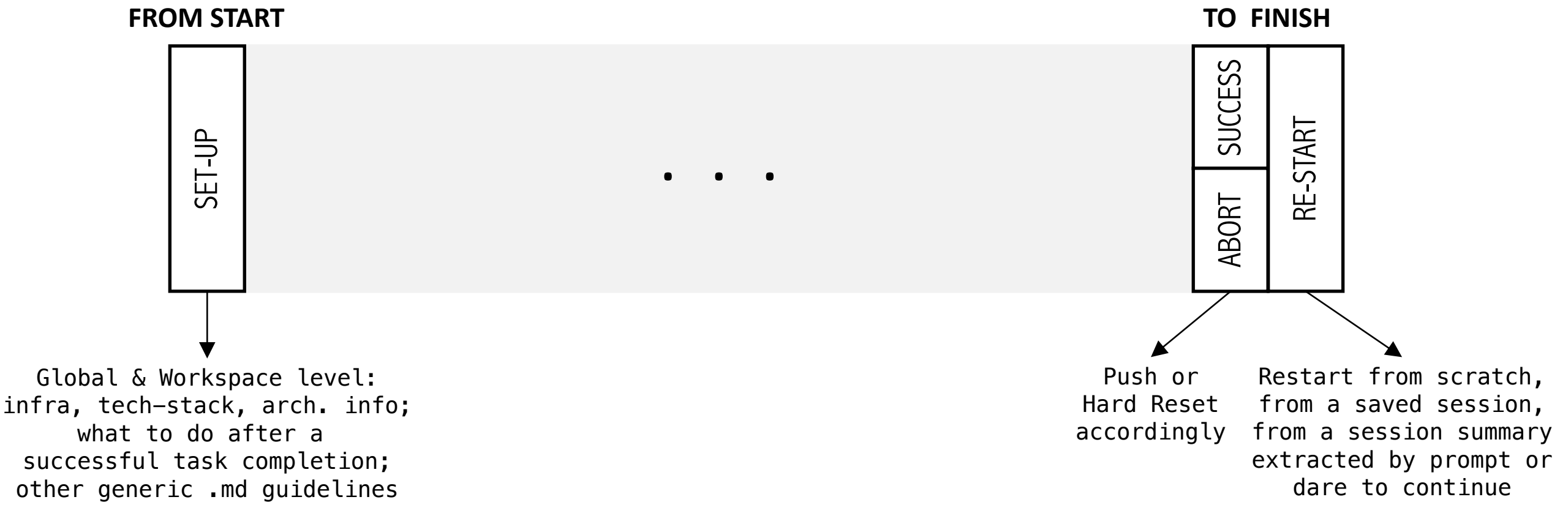
The balance between the agentic **generation** added value and cost of human **verification is** found by tweaking the task size & reviews frequency

1G–1S–1P: HUMAN SETTING UP CONTEXT

These four initial human-driven steps
establish the initial context and task instructions for the Agent

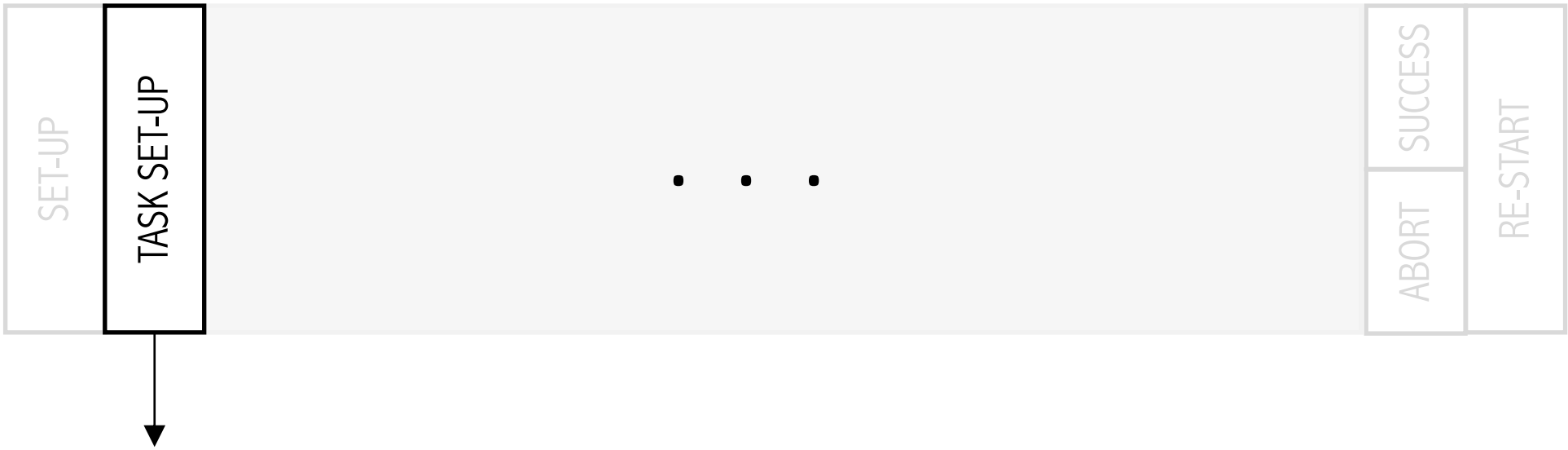
1G–1S–1P: HOW IT BEGINS AND HOW IT ENDS

Initialising the context with the general guidelines and instructions, completing the session before starting a new one



1G–1S–1P: INITIAL HUMAN-DRIVEN SET-UP

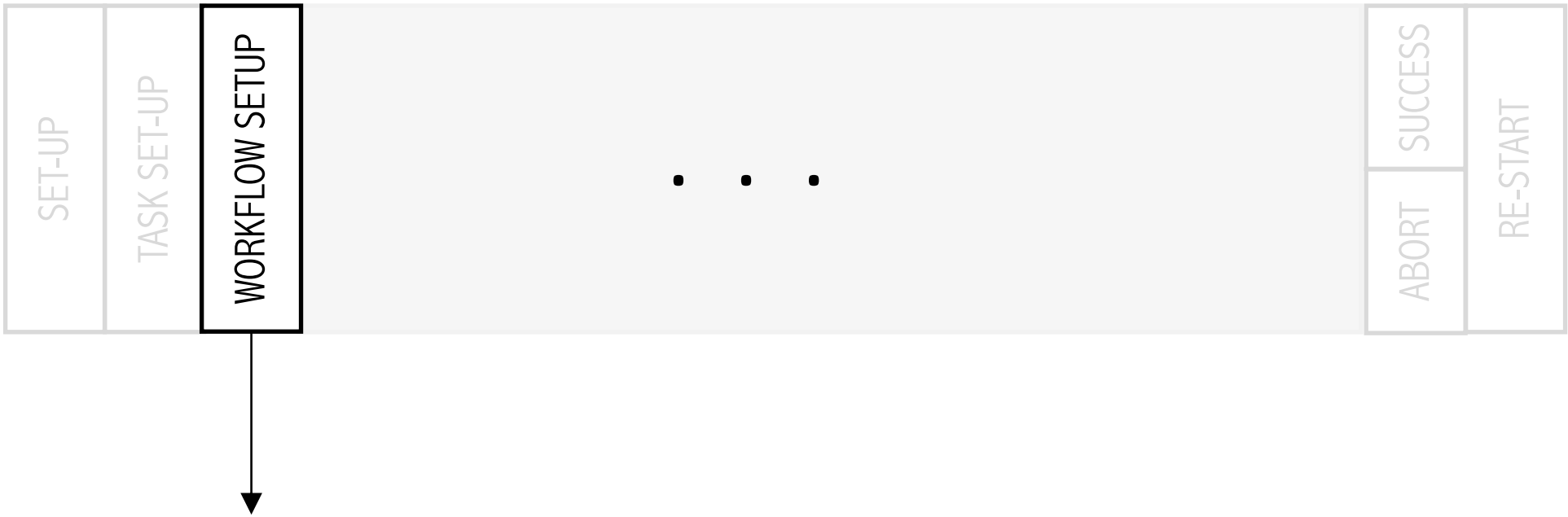
After loading the generic global and project's rules into the context, the task-specific information is added to the context



Task level:
.md guidelines, as for example info on how to execute commands and verify the output for the agent's self-correct & retry loops;

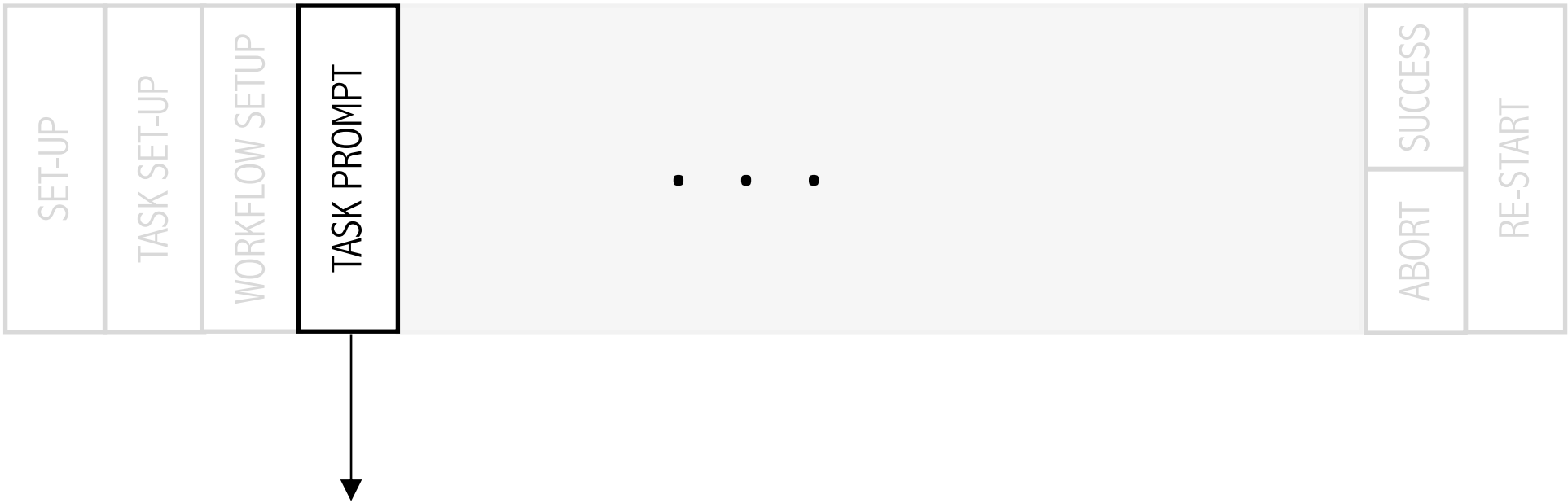
1G–1S–1P: INITIAL HUMAN-DRIVEN SET-UP

The instructions for the current step of the workflow employed (AI-Waterfall, BDD, TDD R-G-R, ATTD, GOOS Outside-In, etc.) are added to the context



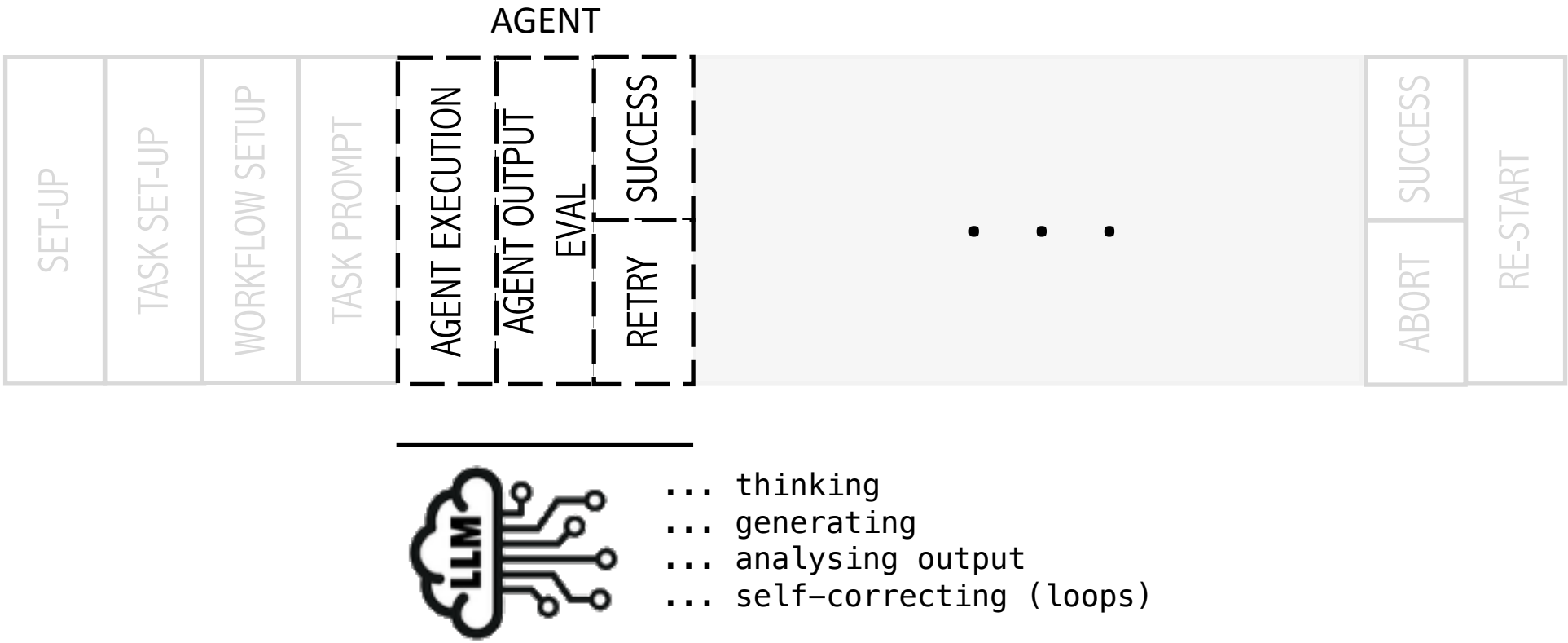
1G–1S–1P: HUMAN-DRIVEN TASK PROMPTING

All task-relevant information *is added to the context*
followed by the prompt with the instructions to complete the task



1G–1S–1P: AGENT GENERATING CODE

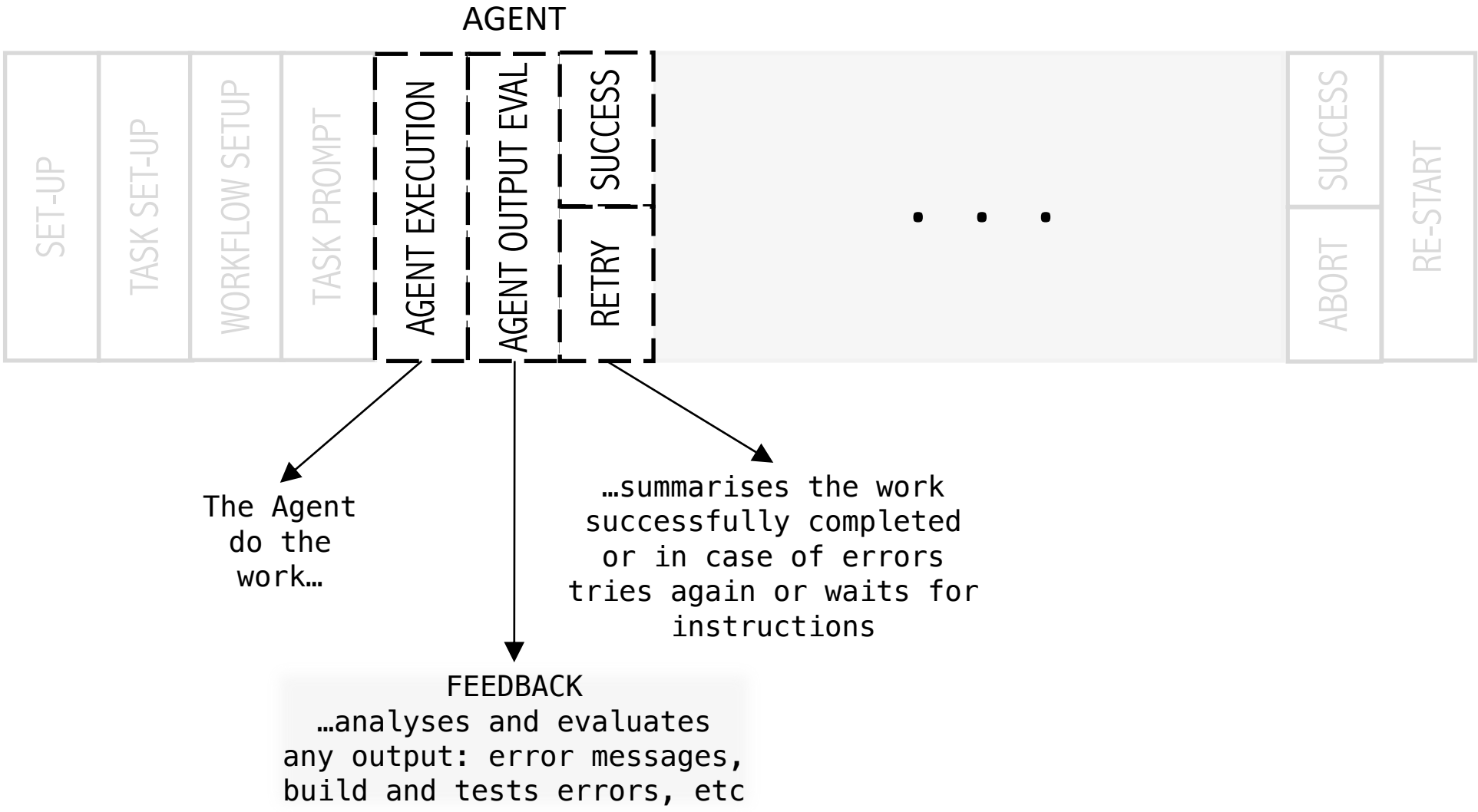
*The **generation**: the agent create the code or what is needed to complete the task and semi-automatically correct the mistakes under human supervision*



The flexibility to switch between various combinations of Terminal CLIs or IDE Chats and different models, even within the same session, accelerated my learning and allowed me to quickly adapt to the rapid evolution of the latest models and tools

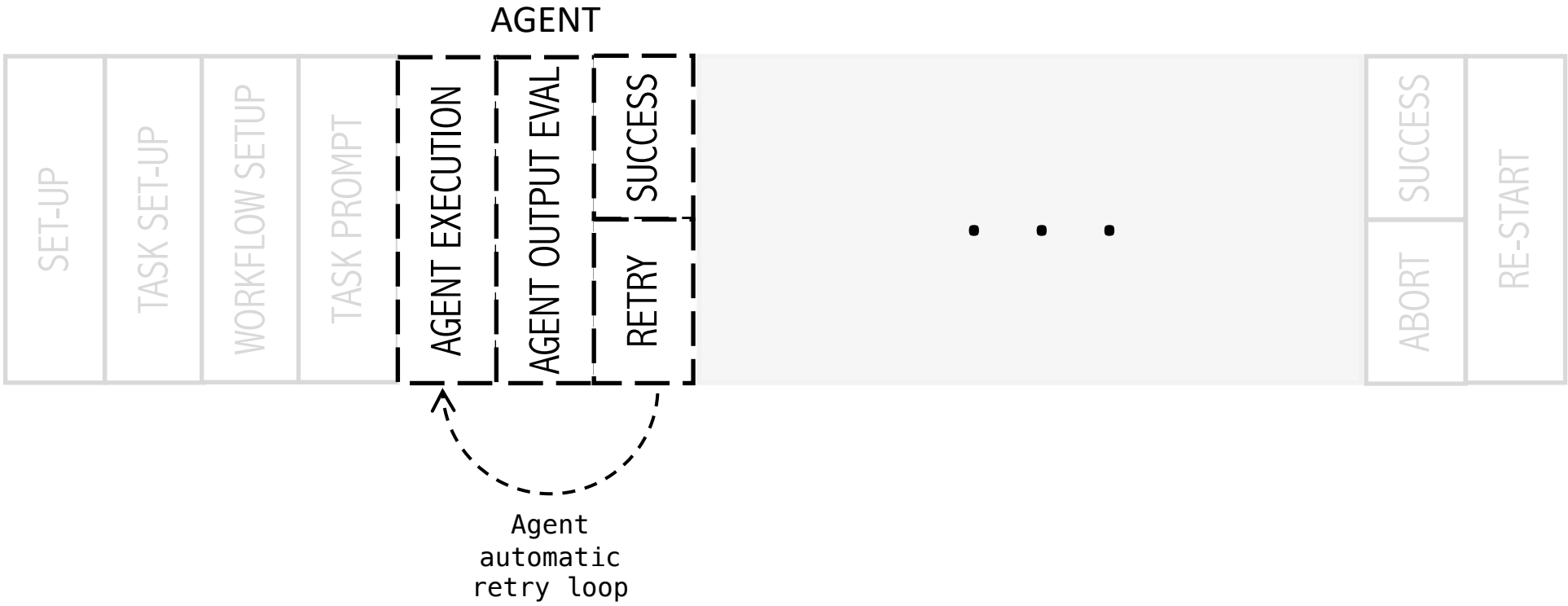
1G–1S–1P: AGENT EXECUTION AND EVALUATION

*The **generation**: the AI executes the task in agentic mode,*
the commands the agent executes produce output that is collected and analysed by the agent



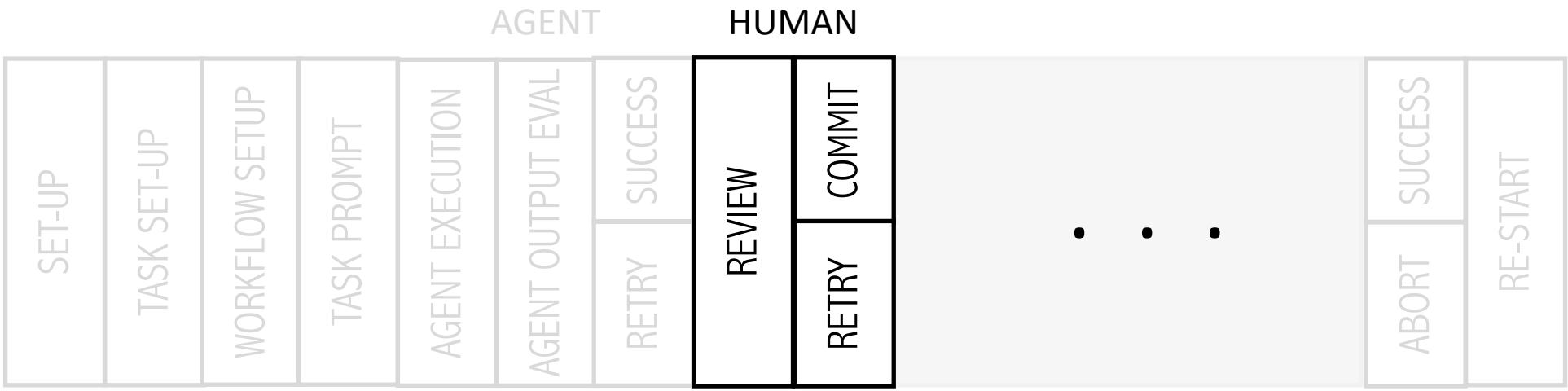
1G–1S–1P: AGENT AUTO-CORRECT FEEDBACK-LOOP

The AI uses the output produced by the failed commands
to try to correct the problem and complete the task successfully



1G–1S–1P: HUMAN REVIEW AND FOLLOW-UP

*The **verification**: developer reviews the results produced by the AI, in order to accept them or instruct the AI to correct them*

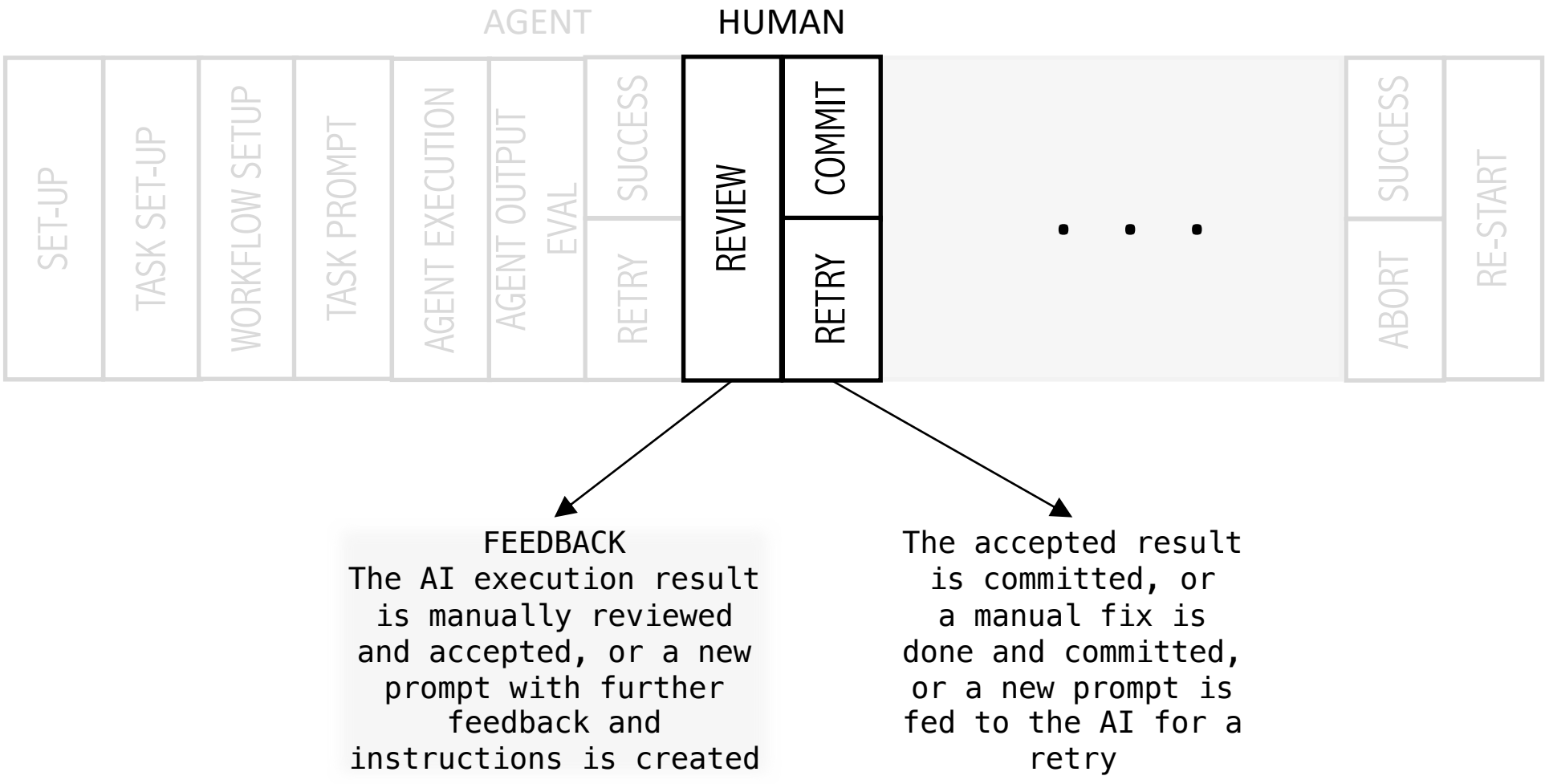


Human review
... reviewing the changes & any errors
... accepting the result or
... manually fixing (if needed)
... prompting the agent for a re-try

To ensure actual productivity gains it is essential to keep the agent on the leash
finding the balance between the agentic **generation** added value and the cost of human **verification**:
here downstream this is done by regulating the frequency and granularity of human supervision

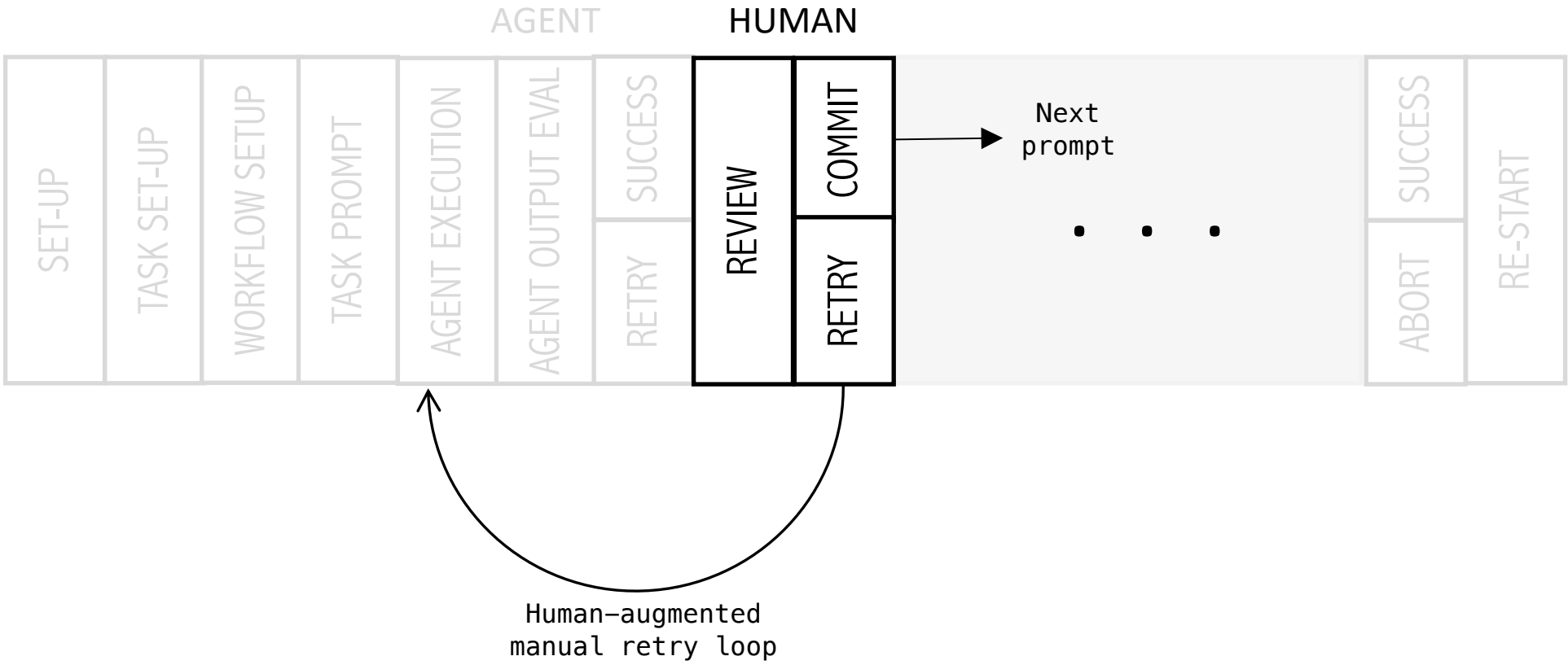
1G–1S–1P: HUMAN REVIEW AND FOLLOW-UP

*The **verification**: developer reviews the results produced by the AI, in order to accept them or instruct the AI to correct them*



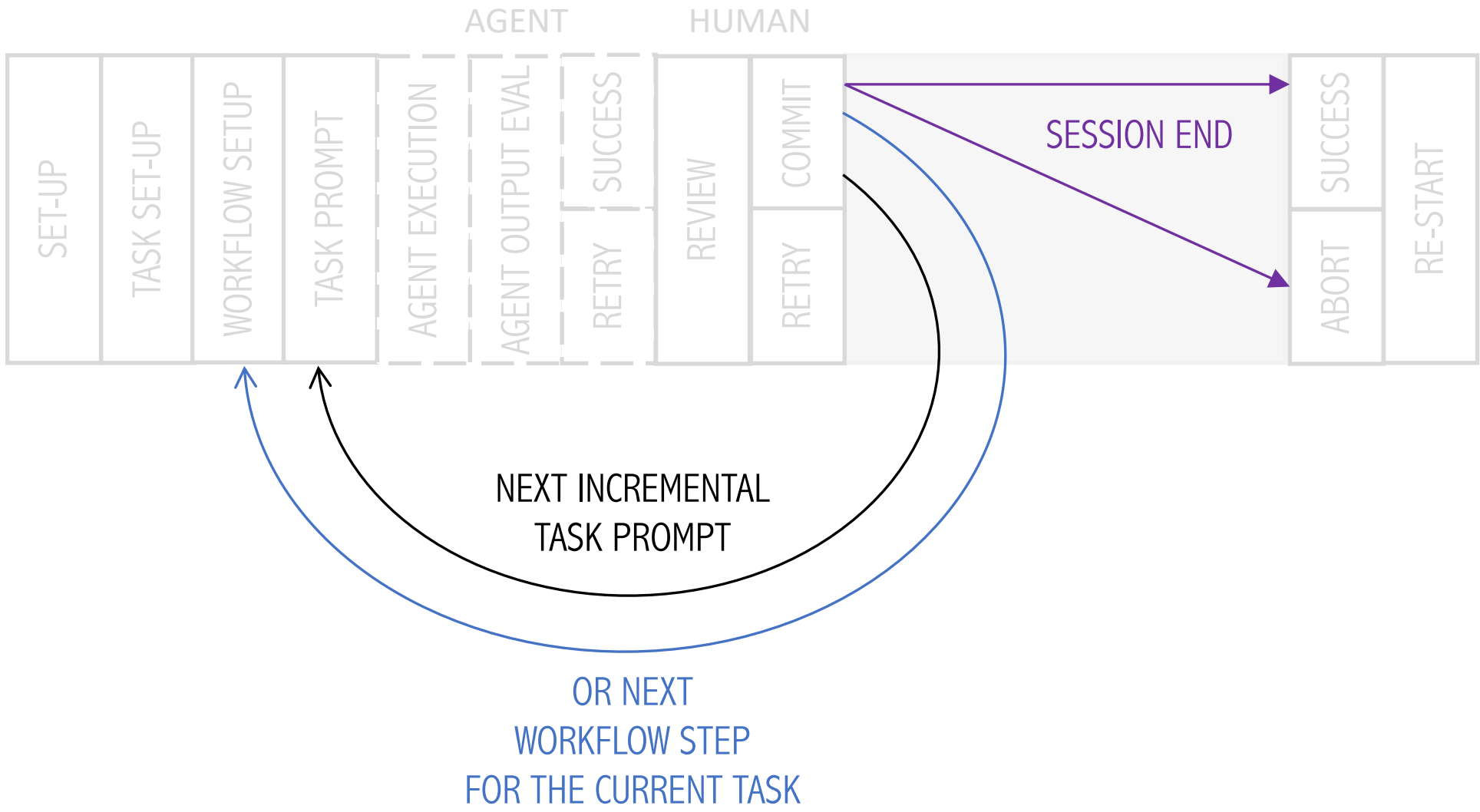
1G–1S–1P: MANUAL FIX FEEDBACK-LOOP

The developer feeds the AI with additional feedback and instructions to fix the problems identified in the results produced by the AI



1G–1S–1P: ITERATING THROUGH THE WORKFLOW & TASK

The task and the workflow proceed iteratively and incrementally until the task is complete and the session ends



1 GOAL – 1 SESSION – 1 PUSH CHOP FLOW

SOME LESSONS LEARNED

Mastery and experience with socio-Tech practices

such as Build/Tests Automation, Continuous Integration and Delivery,
slicing work in small self-contained development cycles a-la eXtreme Programming

serve as invaluable capabilities and critical guardrails for AI-Assisted Coding

---

Enabling autonomous self-correcting Agent's loops makes the agent 10x "smarter"

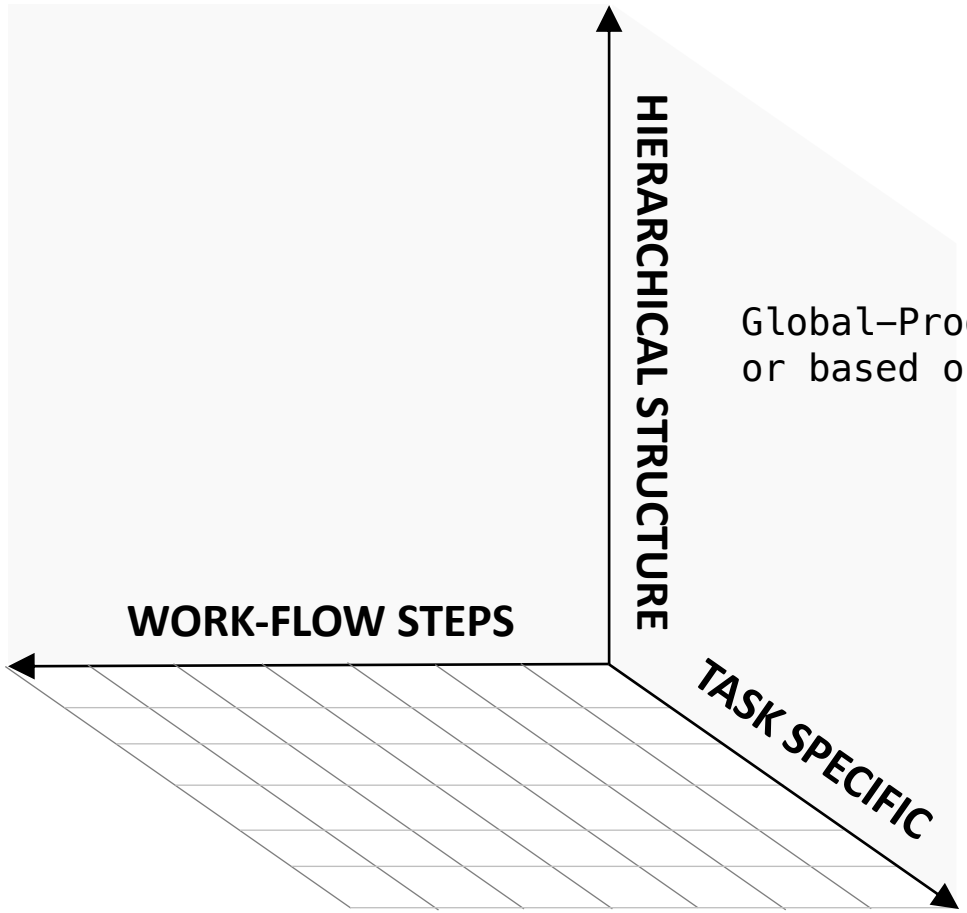
Less is More

when feeding the Context (with .md files, MCPs, etc.)

PROMPTS MODULARISATION AND REUSE

How I'm organising reusable prompts for simple and effective maintenance, evolution

I store reusable prompts in .md files. I am using a text expander to recall them across different CLIs/IDEs/etc
I am organising them along these following three dimensions:



Global–Product–Workspace–Codebase–Sub–module
or based on your µ–svc vs mono–repo codebase

Skills, guidelines and instructions for each step of the dev workflow whatever that is (TDD R–G–R, BDD, etc.)

Architecture, tech–stack, design, domain, code key concepts documentation, task specific skills and guidelines

PROMPTS MODULARISATION AND REUSE

FINAL LESSON LEARNED

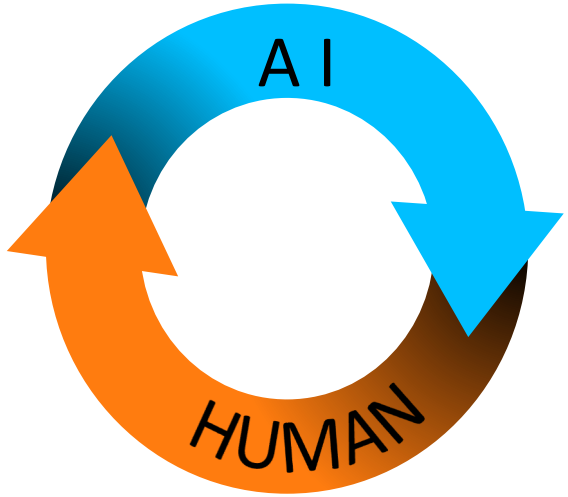
Every AI CLI and IDE offers
its own proprietary way to store and reuse prompts

Text expanders like for example https://espanso.org/
provide an alternative way to store and reuse prompts
across multiple AI CLIs and IDEs
and allow reusable prompts to be organised in files that become part of the code repo

DISCUSSION

Which improvements would you add …

1) To further accelerate the whole **Generation**-**Verification** flow?



1) To support even faster **reuse**, maintenance and evolution of Prompts?